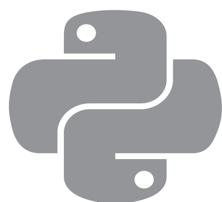


零起点 Python机器学习快速入门

何海群 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

VISIT…

LANZAROTE
Caliente.com

内 容 简 介

本书采用独创的黑箱模式，MBA 案例教学机制，结合一线实战案例，介绍 Sklearn 人工智能模块库和常用的机器学习算法。书中配备大量图表说明，没有枯燥的数学公式，只要懂 Word、Excel，就能够轻松阅读全书，并学习使用书中的知识，分析大数据。本书具有以下特色：

- 独创的黑箱教学模式，全书无任何抽象理论和深奥的数学公式。
- 首次系统化融合 Sklearn 人工智能软件和 Pandas 数据分析软件，不用再直接使用复杂的 Numpy 数学矩阵模块。
- 系统化的 Sklearn 函数 API 接口中文文档，可作为案头工具书随时查阅。
- 基于 Sklearn+Pandas 架构，全程采用 MBA 案例模式，无需任何理论基础，懂 Excel 就可看懂。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

零起点 Python 机器学习快速入门 / 何海群著. —北京：电子工业出版社，2017.5

ISBN 978-7-121-31141-3

Ⅰ. ①零… Ⅱ. ①何… Ⅲ. ①软件工具－程序设计Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2017）第 059262 号

责任编辑：黄爱萍
印　　刷：三河市兴达印务有限公司
装　　订：三河市兴达印务有限公司
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：720×1000　1/16　印张：17.5　字数：252 千字
版　　次：2017 年 5 月第 1 版
印　　次：2017 年 5 月第 1 次印刷
定　　价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前　言

本书是一部意外之作，也是一部惊喜之作。

这是一本非常简单的 Python 机器学习入门教程，具有以下特色。

- 独创的黑箱教学模式，全书无任何抽象理论和深奥的数学公式。
- 首次系统化融合 Sklearn 人工智能软件和 Pandas 数据分析软件，无须使用复杂的 Numpy 数学矩阵模块。
- 三位一体的课件模式：图书+开发平台+成套的教学案例，系统讲解，逐步深入。
- 系统化的 Sklearn 函数 API 接口中文文档，可作为案头工具书随时查阅。
- 基于 Sklearn+Pandas 架构，全程采用 MBA 案例模式，无需任何理论基础，懂 Excel 就可看懂。

本书内容原本是《零起点 Python 足彩大数据与机器学习实盘分析》中的章节，在我们内部小范围使用时，深受学员喜爱，于是，将书中与机器学习相关的内容和案例单独抽取出来，加入部分 Python 入门内容，形成了《零起点 Python 机器学习快速入门》一书。

Python量化三部曲

Python 量化三部曲包括：

- 《零起点 Python 大数据与量化交易》（入门教材）
- 《零起点 Python 量化与机器学习实盘分析》（重点分析 Sklearn）

- 《零起点 Python 量化与 TensorFlow 深度学习实盘分析》（重点分析 TensorFlow）

此外，还有两部补充作品：

- 《零起点 Python 足彩大数据与机器学习实盘分析》
- 《零起点 Python 机器学习快速入门》

Python学习路线

机器学习、人工智能和金融量化基本原理都是相通的，本质上都是数据分析。

本书虽然属于“零起点 Python”系列，但要更好地学习本书，掌握相关的配套程序，最好具备以下基础。

- Python 编程基础，不懂 Python 语言的读者，先花一周时间学习 Python 基本知识以及 Pandas（潘达思）数据分析软件基础操作。
- Top 极宽量化社区有“Python 量化与 zwQuant 学习路线图”，大家可以参考，网址是：http://topquant.vip/forum.php?mod=viewthread&tid=6。
- 先花一周时间学习 Python 基础，再阅读《zwPython 中文手册》，可以少走很多弯路。
- 学好 Python、Pandas 基础后，先将本书通读 1~2 遍。通读时，碰到问题没关系，记录一下跳过去，然后进行精读；正式学习每章的代码时，一定要将代码运行 1~2 遍，培养编程感觉。
- 根据代码学习画流程图，有了流程图就可以把握程序逻辑，重点是程序和策略的逻辑。
- 入门后，多看看配套的课件和程序源码，全套 zwQuant 量化开源程序都带有函数一级的中文注解。
- 在学习过程当中一定要多问，可以在论坛提问，这样大家都受益。

- 本书读者 QQ 群号：124134140。在群文件中有一个小软件，即 Python 流程图绘制软件 CODINGRAPH，使用该软件，拷贝代码就可以自动绘制流程图，对其他编程语言也适用。

网络资源

与本书有关的程序和数据下载，请浏览网站：http://TopQuant.vip，极宽量化社区中的【下载中心】栏目。

本书在 TopQuant.vip 极宽量化社区设有专栏，对本书、人工智能和机器学习有任何建议都可在社区相关栏目发布信息，我们会在第一时间进行反馈和答复。

“零起点Python”系列丛书

本书继续保持了“零起点 Python”系列丛书的一贯风格——简单实用。书中配备了大量的图表说明，没有枯燥的数学公式，只要懂 Word、Excel，就能够轻松阅读全书。

- IT 零起点：无需任何电脑编程基础，只要会打字、会使用 Excel，就能看懂本书，利用本书配套的 Python 软件包，轻松学会如何利用 Python 对股票和足彩数据进行专业分析和量化投资分析。
- 投资零起点：无须购买任何专业软件，本书配套的 zwPython 软件包，采用开源模式，提供 100%全功能、全免费的工业级数据分析平台。
- 配置零起点：所有软件、数据全部采用苹果“开箱即用”模式，绿色版本，无须安装，解压缩后即可直接运行系统。
- 理财零起点：无需任何专业金融背景，采用通俗易懂的语言，配合大量专业的图表和实盘操作案例，轻松掌握各种量化投资策略。
- 数学零起点：全书没有任何复杂的数学公式，只有最基本的加、减、乘、除，轻轻松松就能看懂全书。

致谢

本书的出版要特别感谢电子工业出版社的黄爱萍编辑，感谢她在选题策划和稿件整理方面做出的大量工作。

在本书创作过程中，极宽开源量化团队和培训班的全体成员，也提出过很多宝贵的意见，并对部分内容程序做了中文注解。

特别是吴娜、余勤两位同学，为极宽开源量化文库和 zwQuant 开源量化软件编写文档，并在团队成员管理方面做了大量工作，为他们的付出表示感谢。

何海群（字王）

北京极宽科技有限公司 CTO

2017 年 2 月 25 日

轻松注册成为博文视点社区用户（www.broadview.com.cn），您即可享受以下服务。

- **提交勘误**：您对书中内容的修改意见可在【提交勘误】处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **与我们交流**：在页面下方【读者评论】处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/31141

二维码：



目　录

第 1 章　从阿尔法狗开始说起 1
1.1　阿尔法狗的前世今生 1
1.2　机器学习是什么 2
1.3　机器学习大史记 3
1.4　机器学习经典案例 11

第 2 章　开发环境 13
2.1　数据分析首选 Python 13
2.2　用户运行平台 18
2.3　程序目录结构 19
2.4　Spyder 编辑器界面设置 20
2.5　Python 命令行模式 26
2.6　Notebook 模式 27
2.7　模块库控制面板 29
2.8　使用 pip 更新模块库 33

第 3 章　Python 入门案例 39
3.1　案例 3-1：第一次编程“hello,ziwang” 39
3.2　案例 3-2：增强版“hello,zwiang” 42
3.3　案例 3-3：列举系统模块库清单 44

3.4　案例 3-4：常用绘图风格 45
3.5　案例 3-5：Pandas 常用绘图风格 47
3.6　案例 3-6：常用颜色表 cors 49
3.7　案例源码 50

第 4 章　Python 基本语法 58
4.1　数据类型 58
案例 4-1：基本运算 59
4.2　字符串 61
案例 4-2：字符串入门 61
案例 4-3：字符串常用方法 63
4.3　List 列表 64
案例 4-4：列表操作 65
4.4　Tuple 元组 66
案例 4-5：元组操作 67
4.5　Dictionary 字典 68
案例 4-6：字典操作 68
4.6　数据类型转换 70
案例 4-7：控制语句 71
案例 4-8：函数定义 73
4.7　案例源码 75

第 5 章　Python 人工智能入门与实践 85
5.1　从忘却开始 85
5.2　Iris 经典爱丽丝 89
案例 5-1：Iris 爱丽丝 90
案例 5-2：爱丽丝进化与文本矢量化 92
5.3　AI 操作流程 95

5.4　数据切割函数 98
案例 5-3：Iris 爱丽丝分解 99
案例 5-4：线性回归算法 103
5.5　案例源码 109

第 6 章　机器学习经典算法案例（上） 116
6.1　线性回归 116
6.2　逻辑回归算法 124
案例 6-1：逻辑回归算法 125
6.3　朴素贝叶斯算法 127
案例 6-2：贝叶斯算法 129
6.4　KNN 近邻算法 130
案例 6-3：KNN 近邻算法 133
6.5　随机森林算法 135
案例 6-4：随机森林算法 139
6.6　案例源码 140

第 7 章　机器学习经典算法案例（下） 149
7.1　决策树算法 149
案例 7-1：决策树算法 151
7.2　GBDT 迭代决策树算法 153
案例 7-2：GBDT 迭代决策树算法 154
7.3　SVM 向量机 156
案例 7-3：SVM 向量机算法 157
7.4　SVM-cross 向量机交叉算法 159
案例 7-4：SVM-cross 向量机交叉算法 160
7.5　神经网络算法 161
案例 7-5：MLP 神经网络算法 165
案例 7-6：MLP_reg 神经网络回归算法 168

7.6 案例源码 170

第 8 章 机器学习组合算法 183

8.1 CCPP 数据集 183

案例 8-1：CCPP 数据集 184

案例 8-2：CCPP 数据切割 186

案例 8-3：读取 CCPP 数据集 189

8.2 机器学习统一接口函数 192

案例 8-4：机器学习统一接口 193

案例 8-5：批量调用机器学习算法 201

案例 8-6：一体化调用 205

8.3 模型预制与保存 208

案例 8-7：储存算法模型 210

案例 8-8：批量储存算法模型 213

案例 8-9：批量加载算法模型 215

案例 8-10：机器学习组合算法 219

8.4 案例源码 224

附录 A Sklearn 常用模块和函数 242

附录 B 极宽量化系统模块图 266

1 第 1 章 从阿尔法狗开始说起

1.1 阿尔法狗的前世今生

百度百科的“阿尔法狗”词条是：

> 阿尔法狗（AlphaGo）是一款围棋人工智能程序，由谷歌（Google）旗下 DeepMind 公司的戴密斯·哈萨比斯、大卫·席尔瓦、黄士杰与他们的团队开发，其主要工作原理是“深度学习”。
>
> 2016 年 3 月，该程序与围棋世界冠军、职业九段选手李世石进行人机大战，并以 4:1 的总比分获胜。2016 年年末至 2017 年年初，该程序在中国棋类网站上以“大师”（Master）为注册账号与中国、日本、韩国数十位围棋高手进行快棋对决，连续 60 局无一败绩。不少职业围棋手认为，阿尔法围棋的棋力已经达到甚至超过围棋职业九段水平，在世界职业围棋排名中，其等级分曾经超过排名人类第一的棋手柯洁。
>
> 2017 年 1 月，谷歌 Deep Mind 公司 CEO 哈萨比斯在德国慕尼黑 DLD（数字、生活、设计）创新大会上宣布推出真正 2.0 版本的 AlphaGo。其特点是摈弃了人类棋谱，只靠深度学习的方式成长起来，挑战围棋的极限。

现代社会已经进入后互联网时代，信息资源随手可得，大数据产业风生水起，科技创新层出不穷，不过类似工业革命、卫星登月、原子弹爆炸、Internet 信息高速公路等级别的重大科技突破却一直没有出现，甚至有学者认为，目前的社会处于科技停滞阶段。

直到 2016 年 3 月，谷歌公司的 AlphaGo 横空出世，与围棋世界冠军、职业九段选手李世石进行人机大战，并以 4:1 的总比分获胜。

这个结果震惊了整个社会，特别是学术界人工智能领域的从业人员。过去人们一直认为：围棋是人类智慧的最后堡垒，19×19 的围棋棋盘矩阵可以衍生出天文数字的组合变化，至少在 50 年内，由于计算机技术的限制，人工智能无法达到人类职业选手的标准。没想到，AlphaGo 轻松战胜了人类的围棋冠军。

AlphaGo 程序虽然神秘，但其核心算法却很简单，源自古老的 Monte Carlo 蒙特卡洛算法。

2006 年，欧洲数学家 Rémi Coulomb、Kocsis 和 Szepervari 等学者，在研究围棋程序时，结合蒙特卡洛算法与对手树搜索算法，设计出一种全新的算法：MCTS 蒙特卡罗树搜索算法。

MCTS 全称是 Monte Carlo Tree Search，即蒙特卡罗树搜索算法，是一种在人工智能问题中做出最优决策的方法，它结合了随机模拟的一般性和树搜索的准确性。

而古老的蒙特卡洛算法，正是源自世界上最古老的国际大赌场蒙特卡洛。事实上，现代统计学、博弈学甚至金融领域的量化投资，都不同程度地源自赌场和蒙特卡洛算法。

1.2 机器学习是什么

百度百科的“机器学习”词条是：

> 机器学习（Machine Learning，ML）是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构，使之不断改善自身的性能。它是人工智能的核心，也是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，其主要使用归纳、综合而不是演绎。

Machine Learning（中文版是《计算机科学丛书：机器学习》）是机器学习的经典图书，被卡内基梅隆等许多大学作为机器学习课程的教材，书中主要涵盖了目前机器学习中各种最实用的理论和算法，包括概念学习、决策树、神经网络、贝叶斯学习、基于实例的学习、遗传算法、规则学习、基于解释的学习和增强学习等。

该书作者 Tom Mitchell 在他这本书的序言开场白中给出了一个机器学习的定义。

> 机器学习这门学科所关注的问题是：计算机程序如何随着经验积累自动提高性能。

Marshland 也在 *Machine Learning: An Algorithmic Perspective* 一书的序言中，从算法角度这样描述机器学习。

> 机器学习最有趣的特征之一就是它介于几个不同理论学科之间，主要是计算机科学、统计学、数学和工程学。机器学习经常被作为人工智能的一部分来进行研究，这把它牢牢地置于计算机科学中。若要理解为什么这些算法能够有效工作，则需要一定的统计学和数学头脑，这往往是计算机科学专业的本科生所缺少的能力。

1.3　机器学习大史记

机器学习是人工智能研究较为年轻的分支，它的发展过程大致经历了

三次高潮：

- 20 世纪 40 年代到 60 年代的萌芽期。
- 20 世纪 60 年代末期到 80 年代的摸索期。
- 20 世纪 90 年代到目前的崛起期。

1. 萌芽期

20 世纪 40 年代到 60 年代属于机器学习的萌芽期，以下是期间的一些标志性事件。

（1）最早的人工神经网络原型

1943 年，神经科学家和控制论专家 Warren McCulloch 与逻辑学家 Walter Pitts 基于数理逻辑算法创造了一种神经网络计算模型，这是最早的人工神经网络原型。

（2）Hebb 赫布学习规则

1949 年，心理学家赫布（Donald Hebb）基于神经心理学的学习机制，提出了一种学习假说，即 Hebb 赫布学习规则，开启了机器学习的第一步，

Hebb 赫布学习规则，是一种典型的无监督学习规则，与“条件反射”机理一致，并且已经得到了神经细胞学说的证实。

1948 年，研究人员将这种计算模型的思想应用到了 B 型图灵机上。

1954 年，Farley 和物理学家 Wesley A. Clark 在 MIT 首次用计算机模拟了一个赫布网络。

（3）图灵测试

图灵测试是人工智能哲学方面第一个严肃的提案，图灵测试一词源自阿兰 • 图灵 1950 年的论文《计算机器与智能》。

阿兰 • 图灵（Alan Turing）是英国数学家、逻辑学家，被视为计算机科学之父。1931 年，阿兰 • 图灵进入剑桥大学国王学院，毕业后到美国普

林斯顿大学攻读博士学位。

1950 年，图灵发表了一篇划时代的论文《计算机器与智能》，文中提出了著名的图灵测试：如果一台机器能够与人类展开对话（通过电传设备）而不能被辨别出其机器身份，那么称这台机器具有智能。

2014 年 6 月 8 日，一台计算机（计算机尤金·古斯特曼是一个聊天机器人，一个电脑程序）成功地让人类相信它是一个 13 岁的男孩，成为有史以来首台通过图灵测试的计算机。这被认为是人工智能发展的一个里程碑事件。

（4）第一台神经网络机

20 世纪 50 年代初期，科学家 Walter Pitts 和 Warren McCulloch 是最早的“神经网络”研究学者，他们分析了理想化的人工神经元网络，并且指出了它们进行简单逻辑运算的机制。当时 24 岁的研究生马文 • 闵斯基是他们的学生，1951 年他与 Dean Edmonds 一起建造了第一台神经网络机，称为 SNARC。在接下来的 50 年中，闵斯基是 AI 领域最重要的领导者和创新者之一。

（5）第一批机器人棋手

游戏 AI 一直被认为是评价 AI 进展的一种标准。

1951 年，Christopher Strachey 使用曼彻斯特大学的 Ferranti Mark 1 机器写出了一个西洋跳棋（Checkers）程序；Dietrich Prinz 则写出了一个国际象棋程序。1952 年，IBM 科学家亚瑟 • 塞缪尔开发了一个跳棋程序。该程序能够通过观察当前位置，并学习一个隐含的模型，从而为后续动作提供更好的指导。塞缪尔发现，伴随着该游戏程序运行时间的增加，其可以实现越来越好的后续指导，最终，该程序的棋力甚至可以挑战专业棋手。

通过这个程序，塞缪尔驳倒了普罗维登斯提出的“机器无法超越人类，且无法像人类一样写代码和学习的模式”。他创造了“机器学习”一词，并将它定义为“可以提供计算机能力而无须显式编程的研究领域”。

奇努克（Chinook）是由美国艾尔伯特大学计算机科学家 Jonathan Schaeffer 及其同事开发的一款国际跳棋人工智能程序。1994 年这个程序击败了人类国际跳棋世界冠军马里恩·廷斯利（Marion Tinsley），这是机器程序第一次在竞技游戏中获得官方世界冠军。2007 年这个程序完成了国际跳棋每一步最佳解决方案的信息数据库，成为国际跳棋领域不可能被击败的存在。

（6）逻辑理论家

1955 年，Newell 和 Simon（后来荣获诺贝尔奖）在 J. C. Shaw 的协助下开发了“逻辑理论家（Logic Theorist）”。这个程序能够证明《数学原理》中前 52 个定理中的 38 个，其中某些证明比原著更加新颖和精巧。Simon 认为他们已经“解决了神秘的心/身问题，解释了物质构成的系统如何获得心灵的性质。”（这一断言的哲学立场后来被 John Searle 称为“强人工智能”，即机器可以像人一样具有思想）。

（7）达特矛斯会议

1956 年的达特矛斯会议，是人工智能历史上一个里程碑，这次会议确定了 AI 人工智能的名称和目标，被业界认为是 AI 人工智能诞生的标志。

与会者包括 Ray Solomonoff、Oliver Selfridge、Trenchard More、Arthur Samuel、Newell 和 Simon，他们中的每一位都是未来 AI 领域的殿堂级专家。

会上纽厄尔和西蒙讨论了“逻辑理论家”，而 McCarthy 则说服与会者接受“人工智能”一词作为本领域的名称。

（8）感知机与最小二乘法

1957 年，心理学家罗森·布拉特基于神经感知科学背景提出了第二模型，非常的类似于今天的机器学习模型，它比 Hebb 赫布学习规则更加实用。

基于该模型罗森·布拉特设计出了第一个计算机神经网络——感知机（The Perceptron），它模拟了人脑的运作方式。

3 年后，维德罗首次使用 Delta 学习规则用于感知机的训练步骤，这种方法后来被称为最小二乘方法，这两者的结合创造了一个良好的线性分类器。

2. 摸索期

20 世纪 60 年代末期到 80 年代属于机器学习的摸索期，这段时期，机器学习的发展步伐几乎处于停滞状态。

这主要是因为当时的人工智能领域的理论研究，已经远远领先于当时的硬件水平，独立设备严重滞后，当时的计算机有限的内存和 CPU 运算速度，不足以解决任何实际的 AI 问题。

以下是摸索期的一些标志性事件。

（1）KNN 最近邻算法

1967 年，KNN 最近邻算法（the Nearest Neighbor Algorithm）出现，由此计算机可以进行简单的模式识别。

KNN 最近邻算法，是机器学习当中最简单的方法之一，其设计思想与中国成语“人以类聚、物以群分”类似，KNN 算法的核心思想是，如果一个样本在特征空间中的 k 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别。

（2）BP 和 MLP 神经网络算法

1974 年，伟博斯在博士论文中提出了用误差反向传导来训练人工神经网络，有效解决了异或回路问题，使得训练多层神经网络成为可能。

1981 年，伟博斯在神经网络反向传播（BP）算法中提出 MLP 多层神经网络算法模型。

1985—1986 年神经网络专家鲁梅尔哈特、辛顿、威廉姆斯-赫、尼尔森等人，先后提出了 MLP 与 BP 训练相结合的理念。

至今，BP 算法和 MLP 神经网络算法仍然是主流的机器学习算法。

MLP（Multi-layer Perceptron）多层神经网络算法也称为 MLP 多层感知器，是一种经典的神经网络算法，MLP 算法已被证明是一种通用的函数近似方法，可以被用来拟合复杂的函数或解决分类问题。MLP 在 20 世纪 80 年代曾是相当流行的机器学习方法，拥有广泛的应用场景，比如语音识别、图像识别、机器翻译等，但自 20 世纪 90 年代以来，MLP 遇到来自更为简单的支持向量机的强劲竞争。但由于深层学习的成功，MLP 又重新得到了关注。

BP（Error Back Propagation）算法又被称为误差反向传播算法。BP 算法的基本思想是，学习过程由信号的正向传播与误差的反向传播两个过程组成。由于多层前馈网络的训练经常采用误差反向传播算法，所以人们也经常将多层前馈网络直接称为 BP 网络。

（3）决策树算法

决策树方法最早产生于 20 世纪 60 年代，是一种逼近离散函数值的方法。它是一种典型的分类方法，首先对数据进行处理，利用归纳算法生成可读的规则和决策树，然后使用决策对新数据进行分析。本质上决策树是通过一系列规则对数据进行分类的过程。

1986 年，人工智能专家 J.Ross Quinlan 提出了著名的 ID3 算法，可以减少树的深度，大大加快运行速度。

后来 C4.5 算法在 ID3 算法的基础上进行了较大改进，使决策树算法既适合于分类问题，又适合于回归问题。

国际权威的学术组织——数据挖掘国际会议 ICDM（the IEEE International Conference on Data Mining），在 2006 年 12 月评选出了数据挖掘领域的十大经典算法，C4.5 算法排名第一。C4.5 算法是机器学习算法中的一种分类决策树算法，其核心算法是 ID3 算法。

3. 崛起期

20 世纪 90 年代到目前，属于人工智能、机器学习的崛起期，崛起的最主要的两个原因是：

- 计算机硬件飞速发展，特别是 GPU 图形运算加速卡，以及计算机集群技术的发展，使电脑的运算比 20 世纪 80 年代提高了数万倍。
- 互联网产业的普及与推广，网络上面沉淀了海量的数字化资源，大数据产业的崛起，为机器学习提供了足够的研究和分析素材。

以下是这段时期的一些标志性事件。

（1）Boosting 算法

Boosting 是一种提高任意给定学习算法准确度的方法，其思想起源于 Valiant 提出的 PAC（Probably Approximately Correct）学习模型。

Valiant 和 Kearns 提出了弱学习和强学习的概念，识别错误率小于 1/2，即准确率仅比随机猜测略高的学习算法，称为弱学习算法；识别准确率很高，并能在较短时间内完成的学习算法，称为强学习算法。

1990 年，Schapire 构造出一种多项式级的算法，对该问题做了肯定的证明，这就是最初的 Boosting 算法。一年后，Freund 提出了一种效率更高的 Boosting 算法。但是这两种算法存在共同的实践上的缺陷，那就是都要求事先知道弱学习算法正确的下限。

1995 年，Freund 和 Schapire 改进了 Boosting 算法，提出了 AdaBoost（Adaptive Boosting）算法，更容易应用到实际问题当中。之后，Freund 和 Schapire 进一步提出了改变 Boosting 投票权重的 AdaBoost. M1、AdaBoost. M2 等算法，在机器学习领域受到了极大的关注。

（2）SVM 向量机算法

1995 年，瓦普尼克和科尔特斯提出 SVM（Support Vector Machines）向量机算法，也称为 SVM 支持向量机算法。

SVM 向量机算法是一个有监督的集成决策树算法模型，通常用来进行模式识别、分类以及回归分析，支持向量机的提出，有很深的数学理论背景。

SVM 向量机算法是近年来机器学习领域中一个最重要的突破，从此将机器学习分为神经网络算法和 SVM 向量机算法两大流派。

SVM 向量机算法应用核函数的展开定理，无须知道非线性映射的显式表达式。由于是在高维特征空间中建立线性学习机，所以与线性模型相比，不但几乎不增加计算的复杂性，而且在某种程度上避免了“维数灾难”。

神经网络算法，特别是深度学习算法，能够实现更加艰难的任务，如目标识别、语音识别、自然语言处理等，但这绝对不意味着其他机器学习方法的终结。

尽管深度学习的成功案例迅速增长，但是对这些模型的训练成本是相当高的，调整外部参数也很麻烦。同时，SVM 向量机算法的简单性促使其成为广泛使用的机器学习方式。

（3）深蓝人机国际象棋大赛

1997 年 5 月 11 日，对于人工智能行业而言，可以说是历史性的一天，IBM 的“深蓝”电脑在国际象棋比赛中，首次击败了等级分排名世界第一的棋手加里·卡斯帕罗夫。蓝电脑的胜利标志着国际象棋和人工智能的新时代。

“深蓝”是美国 IBM 公司生产的一台超级国际象棋电脑，重 1270 公斤，有 32 个“大脑”（微处理器），每秒钟可以计算 2 亿步。深蓝输入了 100 多年来优秀棋手的 200 多万对局。

人与计算机的首次国际象棋对抗赛是在 1963 年，国际象棋大师兼教练大卫·布龙斯坦怀疑计算机的创造性能力，同意用自己的智慧与计算机较量。

1996 年 2 月 10 日，超级电脑“深蓝”首次挑战西洋棋世界冠军卡斯帕罗夫，但以 2∶4 落败。

（4）RF 随机森林算法

随机森林算法，英文名称 Random Forest，简称 RF 算法，是一种集成决策树模型，由布雷曼博士在 2001 年提出，它是由一个随机子集的实例组成，并且每个节点都是从一系列随机子集中选择，且由于这个性质被称为随机森林（RF）。其算法在理论和经验上证明了该算法对过拟合的抵抗性，是一种非常稳健的机器学习算法模型。

（5）Deep Learning 深度学习

2006 年，神经网络专家 Hinton 提出了神经网络 Deep Learning 深度学习算法，使神经网络的能力大大提高，向 SVM 支持向量机发出挑战。

当前机器学习领域最热门的两大算法，是 Deep Learning 深度学习算法和 SVM 向量机算法，它们都是给予统计学的机器学习方法。

深度学习的概念源于神经网络的研究，是机器学习研究中的一个新的领域，其动机在于建立、模拟人脑进行分析学习的神经网络，模仿人脑的机制来解释数据，例如图像、声音和文本。

（5）AlphaGo 围棋大师

阿尔法狗 AlphaGo，是一款围棋人工智能程序，由谷歌（Google）旗下 DeepMind 公司的戴密斯 • 哈萨比斯、大卫 • 席尔瓦、黄士杰与他们的团队开发，其主要工作原理是“深度学习”。

关于阿尔法狗 AlphaGo 的更多资料，请参见前文或浏览相关网页，在此不再重复。

1.4　机器学习经典案例

目前人工智能、机器学习正处于黄金时期，各种应用随处可见，以下是一些常见的机器学习应用案例。

- 机器人客服：当你拨打移动、银行等公司的服务热线时，大部分都是通过人工智能技术合成的电脑客服在和你沟通，只有遇到电脑客服无法处理的问题，才会转给人工处理。
- 语音助理：苹果的 Siri 和微软的小娜，就是采用人工智能开发的语音助手，两者会自动识别你的语音命令。
- 垃圾邮件检测：根据邮箱中的邮件，识别哪些是垃圾邮件，哪些不是。这样的模型可以使用程序帮助归类垃圾邮件和非垃圾邮件。
- 信用卡欺诈检测：根据用户的信用卡交易记录，识别哪些交易是该用户操作的，哪些不是。
- 数字识别：根据信封上手写的邮编，识别出每一个手写字符所代表的数字。这样的模型可以帮助程序阅读和理解手写邮编，并根据地理位置分类信件。
- 人脸识别：根据相册中的众多数码照片，识别出包含某一个人的照片，根据人脸管理照片。
- 产品推荐：根据用户的购物记录和冗长的收藏清单，识别出这其中哪些是该用户真正感兴趣，并且愿意购买的产品。京东、淘宝都有大量这样的案例。
- 医学分析：根据病人的症状和病人资料数据库，预测该病人可能患了什么病。
- 股票交易：这个就是目前非常火热的智能投顾技术，根据股票现在和以往的价格波动，判断这支股票是应该建仓、持仓还是减仓。
- 客户细分：根据用户在产品试用期的行为模式和所有用户过去的行为，识别出哪些用户会转变成该产品的付款用户，哪些不会。
- 形状鉴定：根据用户在触摸屏幕上的手绘和一个已知的形状资料库，判断用户想描绘的形状。
- 艺术作品鉴别：艺术作品的鉴别是极其复杂的，专家通常会判断它所属的类型、流派、作者和时代。新泽西州罗格斯大学的专家使用人工智能技术，鉴别画作的流派和作者。
- 婚恋匹配：国内外的婚恋交友网站经常通过性格测试来提取数据，然后通过人工智能分析，进行婚恋匹配。

2 第 2 章 开发环境

2.1 数据分析首选Python

Python 人工智能和机器学习在本质上底层都是数据分析，而对于数据分析，行业首选的编程工具就是 Python 语言。

经常有网友认为，自己投资失败或者量化策略失效是因为 Python 开发环境不好，希望自己重新设计一个，甚至希望用 C、Java 来重新开发量化软件。

对于提出这类问题的网友，笔者曾经这样答复：“量化的核心是策略，至于量化软件的运行速度、用户 GUI 界面都是程序员的事情。大家是做金融操盘手，只要会写策略就行。就像会计，如果觉得 Excel、用友或金蝶财务软件的某些功能不好用，另外再选一套财务软件就好，完全没必要自己也去写一套财务软件。”

在软件工程领域有一句著名的格言：“Don't Reinvent the Wheel”——不要重复发明轮子。

这句话说得非常经典，也非常有道理。

至于为什么选择 Python 语言作为数据分析和量化分析的首选开发平台，大家可以参考 Top 极宽量化社区的介绍：《zwPython 史前故事》。

金融股票、足彩分析。机器学习等量化分析系统，本质上都是数据分析，如果大家有兴趣，还可以浏览笔者的博客内容《大数据，why Python》：http://ziwang.com/forum. php?mod=viewthread&tid=58。

1. 入门简单，功能强大

Python 是最适合初学者的编程语言，也是目前 IT 行业唯一的入门简单、功能强大的工业级开发平台。

事实上，Python 已经超越普通编程语言，成为 IT 行业的万能开发平台。

（1）入门简单

任何熟悉 JavaScript 脚本、Visual Basic、C 语言、Delphi 的用户，通常一天即可学会 Python。

即使是不会编程的美工设计师、打字员，一周内也能熟练掌握 Python，学习难度绝对不会高于 Photoshop、五笔，至少笔者和许多程序员一样，现在还不会使用五笔字型。

zwPython 原本就是面向美工设计的，最早的用户手册名字就叫《设计师学编程》。

（2）功能强大

海量级的 Python 模块库，提供了 IT 行业最前沿的开发功能。

- 大数据：Pandas 已经逐步碾压 R 语言。
- CUDA：高性能计算，Python、与 C（C++）、Fortran 是 NVIDIA 公司官方认可的 3 种编程语言，也是目前唯一适合大众用户的 CUDA 编程工具。
- 机器学习：TensorFlow、PyTorch、Scikit-learn、Theano 都是国际上热门的机器学习平台。

- 自然语言：NLTK，全球首选的自然语言处理平台；SpaCy，工业级 NLP 平台。
- 人脸识别：OpenCV，光流算法、图像匹配、人脸算法，简单优雅。
- 游戏开发：Pygame 提供图像、音频、视频、手柄、AI 等全套游戏开发模块库。
- 字体设计：Fontforge，是唯一商业级的字体设计开源软件，内置的脚本语言和底层核心的 Fonttools，都是基于 Python。
- 电脑设计：blend、GIMP、inkscape、MAYA、3DS 都内置或扩展了 Python 语言支持。

上述 Pandas、CUDA、TensorFlow、PyTorch、Scikit-learn、Theano 为 Python 模块库或 IT 行业术语。

目前热门的 iOS、安卓等手机 APP 应用开发，也可以使用 Python 开发，但基本都是商业收费模块，因此未集成到 zwPython 软件包，大家可以自行搜索。

既然 Python 如此美好，而且是 100%免费的开源软件，学习 Python 的人也越来越多，为什么 Python 相对于 C#、JavaScript、Visual Basic、C 语言，始终还只是一种小众语言呢？

笔者认为，Python 的“大众化”之路，存在以下两个瓶颈。

- 配置：软件行业有一句俗话“搞懂了软件配置，就学会了一半”。对于 Python 和 Linux 等许多开源项目而言，80%的问题都出现在配置方面，尤其是模块库的配置。
- OOP（面向对象程序设计）：大部分人都认为 Python 是一种“面向对象”的编程语言，而 OOP 的编程风格，业界公认比较繁杂。

如果能够解决好以上两个问题，Python 的学习难度可以降低 90%，而在应用领域和开发效率方面，则可以瞬间提升数十倍效能，而且这种提升是零成本的。

2. 难度降低，性能提高

笔者在 WinPython 软件包的基础上，推出了“zwPython”——字王集成式 Python 开发平台。

- 业界首次提出“零配置、零对象”的研发理念，绿色软件封装模式，类似 MAC 开箱即用风格，无须安装，解压即可直接使用，还可以放入 U 盘，支持 Mob-APP 移动式开发编程。
- “外挂”式“核弹”级开发功能，内置很多功能强大、IT 前沿的开发模块库，例如 OpenCV 视觉、人脸识别、CUDA 高性能 GPU 并行计算（Opencl）、Pandas 大数据分析、TensorFlow、PyTorch 机器学习、NLTK 自然语言处理。
- 便于扩展，用户可以轻松增删相关模块库，全程智能配置，无须用户干预，就好像拷贝文件一样简单，而且支持 U 盘移动便携模式，真正实现了“一次安装，随处可用”。
- 针对中文开发文档缺乏、零散的不足，内置多部中文版 Opencv、Fontforge 和 Python 入门教材。
- 大量示例脚本源码，涵盖 OpenCV、CUDA、Opencl、Pygame 等。

如此种种只是为了便于 IT 行业外的用户能够“零起步”、快速入门，并且短时间内应用到生产环节当中去。

zwPython 前身是 zw2015sdk：字王智能字模设计平台，原设计目标是向广大美工设计师提供一款统一的、可编程的字体设计平台，以便于大家交流。

- 美工设计师、美工都是文艺青年、IT 小白，所以，简单是必需的，开箱即用也必须是标配。
- 做设计，图像处理 PIL、Matplotlib 模块是必需的。
- 集成了 Opencv 作为图像处理、匹配模块，自然也提供了机器学习功能。
- 字模处理数据量很大，属于大数据范畴，须集成 SciPy、NumPy 和 Pandas 数据分析模块。

- 由于原生 Python 速度慢，所以增加了 Pycuda、Opencl 高性能 GPU 计算模块。

如此一而再、再而三地扩充，发现 zwPython 已经基本覆盖了目前 Python 和 IT 编程 90%的应用领域，因此又增加了部分模块，将 zwPython 扩展成为一个通用的、集成式 Python 开发平台。

3.“零对象”编程模式

虽然很多人认为 Python 是一种“面向对象”的编程语言。

但对于初学者而言，把 Python 视为一种 Basic 风格的、过程式入门语言，学习难度可以降低 90%，基本上学习一小时即可动手编写学习代码。

有人说，“面向对象”最大的好处是方便把人脑子搅乱。

Windows、Linux、UNIX、Mac OSX 内核都是使用 C 语言、汇编写的。有一种系统是 C++写的内核，就是诺基亚的塞班系统，据说代码量比 Windows XP 还大，连他们自己的程序员都无法维护，最后就死掉了。

简而言之，“面向对象”的代码风格：一个字“繁”、两个字“繁繁”、三个字“繁繁繁”。

“零对象编程模式，用 Basic 的方式学习 Python”，是笔者向 Python 等编程语言的入门用户提出的一种全新的学习理论，一家之言，仅供参考。

Talk is cheap,Show me the code！大家还是多多动手。

大家很容易理解“零配置”，下面关于“零对象”再补充几点。

- 不写“面向对象”风格的代码不等于不能使用，对于各种采用“对象”模式开发的模块库，我们仍然可以直接调用。
- 将 Python 视为非“面向对象”语言并非大逆不道，事实上，许多人认为，Python 也是一种类似 LISP 的“函数”编程语言。
- 笔者从事编程十多年，从未用过“面向对象”模式编写过一行“class”（类对象）代码，依然可以应对各种编程工作。

- 目前“面向对象”编程理论，在业界仍然争论不休，入门者功力不够，最好避开强者之间的火力杀伤。
- “面向对象”的鼻祖 C++ 11 标准，直到 2015 年依然处于推广阶段，而且争议纷纷。
- “面向对象”过于复杂，与“人生苦短，我用 Python”的优雅风格天生不合。

2.2 用户运行平台

本节主要讲解 Python 开发环境和数据包的配置、应用流程方面的知识。

本书所有案例程序均采用纯 Python 语言开发，除特别指明外，均默认使用 Python 3 语法，且经过 zwPython 平台测试。

zwPython 是 TopQuant 极宽公司推出的一个 Python 集成版本，功能强大，属于免费开源软件。系统内置了数百种专业的 Python 模块库，无须安装，解压即用。有关 zwPython 的使用，可参考软件自带的《zwPython 用户手册》。

本书所有案例程序可用于 zwPython 平台，以及各种支持 Python 3 的设备平台，包括 Linux 操作系统、Mac 苹果电脑，以及安卓系统、树莓派。

其他非 zwPython 用户运行本书程序时，如果出现问题，通常是缺少有关的 Python 模块库，可以根据调试信息安装相关的 Python 模块库，再运行相关程序。

zwPython 及本书配套下载地址，请参见 Top 极宽量化社区“下载中心”：

http://topquant.vip 或 http: //ziwang.com。

2.3　程序目录结构

本书配套程序的工作目录是 zwPython\py_demo，这也是本书默认的工作目录，凡是没有特别标注目录的脚本文件，一般都位于该目录。有关的程序会定时在读者群发布更新，请读者及时下载。

相比普通的 Python 版本，本书配套的教学版的 zwPython 目录中多了一个 py_demo 目录。

py_demo 目录收录了相关培训课程的配套代码和所需数据，py_demo 目录也可以复制到其他目录，建议放到 zwPython 根目录下。

zwPython 目录结构中的其他子目录如下。

- \zwPython\doc\：用户文档中心，包括用户手册和部分中文版的模块库资料。
- \zwPython\py35\：Python 3.5 版本系统目录，除增加、删除模块库外，一般不需要改动本目录下的文件，以免出错。另外，如果日后 Python 版本升级，这个目录也会变化，如 Python 3.6，会采用 py36 作为目录。
- \zwPython\demo\：示例脚本源码。
- \zwPython\zwrk\：zw 工作目录，用户编写的脚本代码文件建议放在本目录下。
- \zwPython\TFB\：Top Football，又称 Top Quant for Football，简称 TFB，极宽足彩量化分析系统（快速学习版无此目录）。

足彩版本，配套的 TFB 极宽足彩量化分析系统，与 zwPython 进行了集成处理，可直接输入使用，支持 Python 3.x。移植时或使用其他 Python 环境时，可以把 TFB 目录下的脚本文件全部复制到自己的代码工作目录，注意代码里面有关数据文件目录的设置。

2.4 Spyder编辑器界面设置

1. 开发环境界面设置

在设置界面之前，可随意把一个 Python 源码文件，用鼠标拖到 Python 编程语言编辑器 Spyder 的编辑框中，如图 2-1 所示。

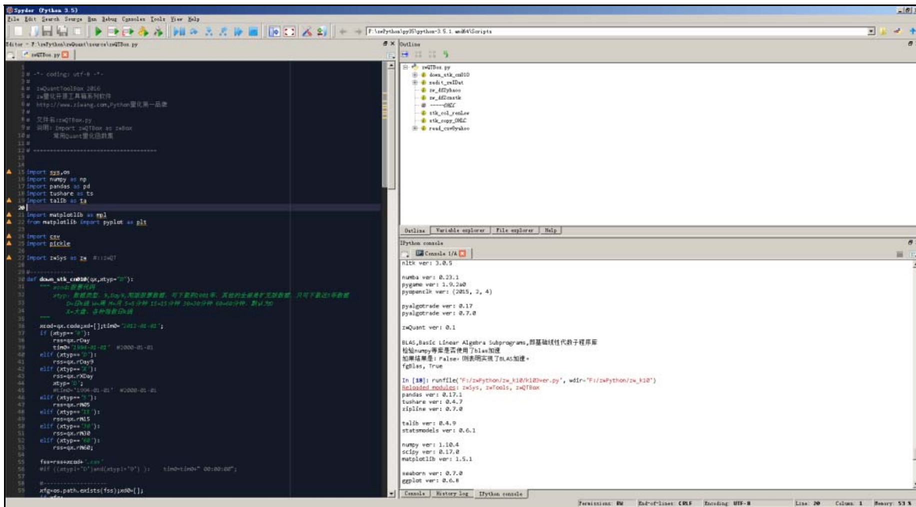


图 2-1 Python 编程语言编辑器 Spyder 编辑框界面

Spyder 编辑器的界面设计非常合理，参考了 MATLAB，特别适合量化分析，国际投行一般都选择用这种布局作为标配。

通常需要优化的只有 Outline（导航）面板，又称函数列表面板，类似 Delphi 语言的 Struct 函数列表面板。

在 Spyder 编辑器默认配置中，Outline 面板是不显示的，单击菜单 View→Panes→Outlines，如图 2-2 所示，将显示 Outline 面板。

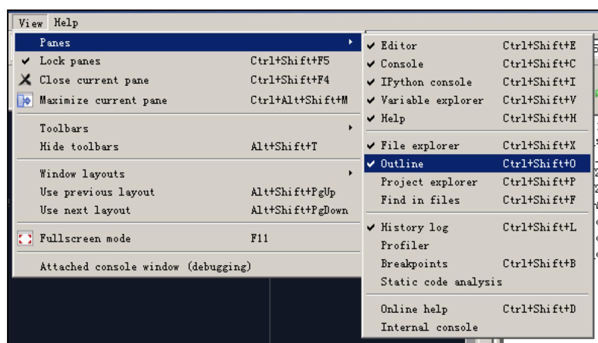


图 2-2　显示 Outline 面板

Outline 面板显示后，它的默认位置在代码编辑器和右侧窗口的中间。

建议单击 Outline 面板左上角的“窗口缩放”按钮，拖动面板到右上方，将其与 Var（变量）面板、File（文件）面板等合并。

Outline 面板的作用是对代码中的函数、类、变量进行快速导航定位。单击 Outline 面板的函数、类、变量名称后，左侧代码编辑器就会自动移动到相关代码，如图 2-3 所示。对于大型项目而言，使用 Outline 面板可以提高效率。

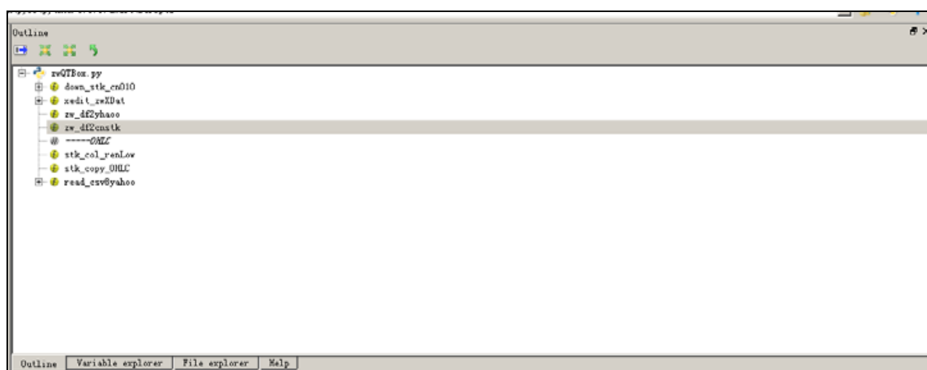


图 2-3　Spyder 编辑器 Outline 面板

需要注意的是，由于 Spyder 软件未来版本升级，具体操作界面和细节可能会有所不同，本书其他软件和模块也是如此，这个属于正常情况，大家无须担心。

2. 代码配色技巧

zwPython 的 IDE 代码编辑器是 Spyder，默认配色是 Spyder 模式，采用白底黑字，与传统的 IDE 环境差别很大，如图 2-4 所示。

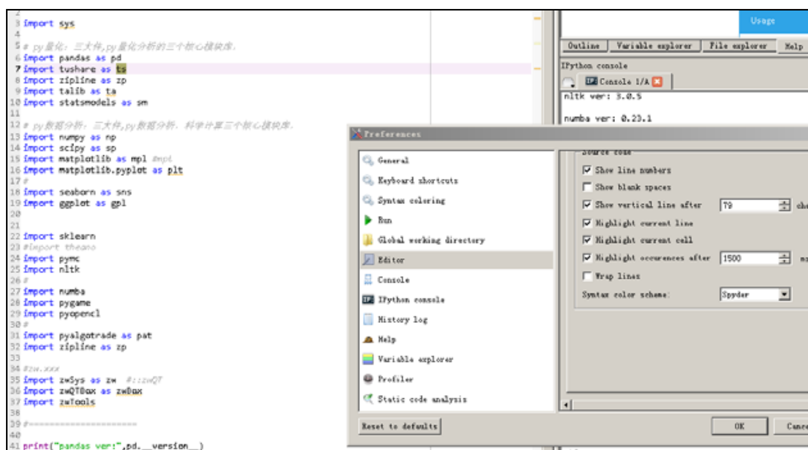


图 2-4　Spyder 编辑器配色模式

如图 2-5 所示是最新的 delphi-xe10 的编辑器配色模式（Twilight 模式）。

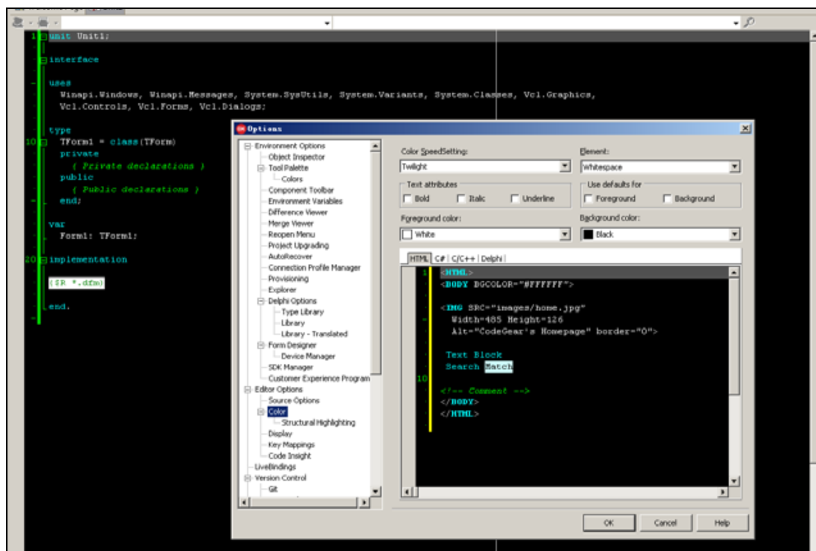


图 2-5　delphi-xe10 编辑器配色模式

这种黑底模式也是微软 VS 等开发平台标准的代码编辑器配色模式。幸运的是，Spyder 编辑器内置的配色模式里也有类似的模式。

运行 Spyder 编辑器，单击菜单 Tools→Preferences，打开 Preferences 对话框。在左侧的列表框中选择 Editor（编辑器），在右侧 Display（显示）面板的 Syntax color scheme（语法配色方案）下拉列表框中选择 Spyder/Dark（暗调）模式即可。如图 2-6 所示。不同版本的 Spyder 编辑器调整细节会有所不同，请读者注意。

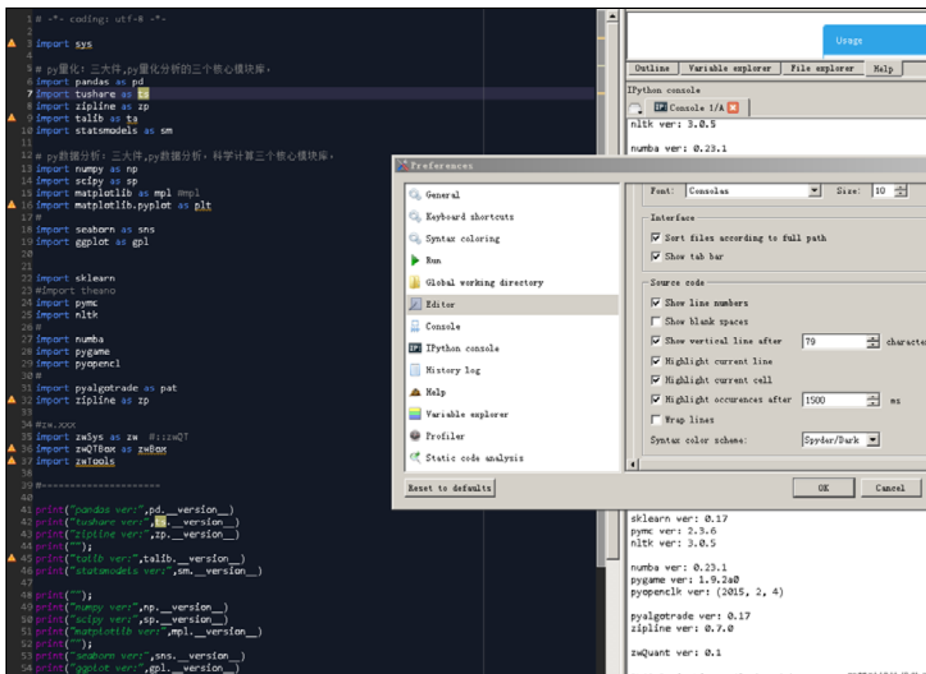


图 2-6　整 Spyder 编辑器配色模式

3. 图像显示配置

Python 语言的 Spyder 编辑器默认的图像显示尺寸，对于高清显示器来说有些偏小，需要进行调整，具体步骤如下。

（1）单击菜单 Tools→Preferences，打开 Preferences 对话框。

（2）单击左侧列表框中的 IPython console（IPython 控制台）。

（3）在对话框的右侧选择 Graphics 选项卡。

（4）在 Graphics backend 选项区中，Backend 选项默认为 Inline，一般不需要改，如要进行交互分析，可以设置为 Automatic（自动模式）或者 Qt（Qt 模式）。

（5）在 Inline backend 选项区中可以调整内置图像的大小，默认值 Width 为 8、Height 为 5，建议将 Width 改为 16、Height 改为 8。

此外，建议勾选对话框上部的 Automatically load Pylab and NumPy modules 复选框（会自动加载 Pylab、NumPy 模块），如图 2-7 所示。

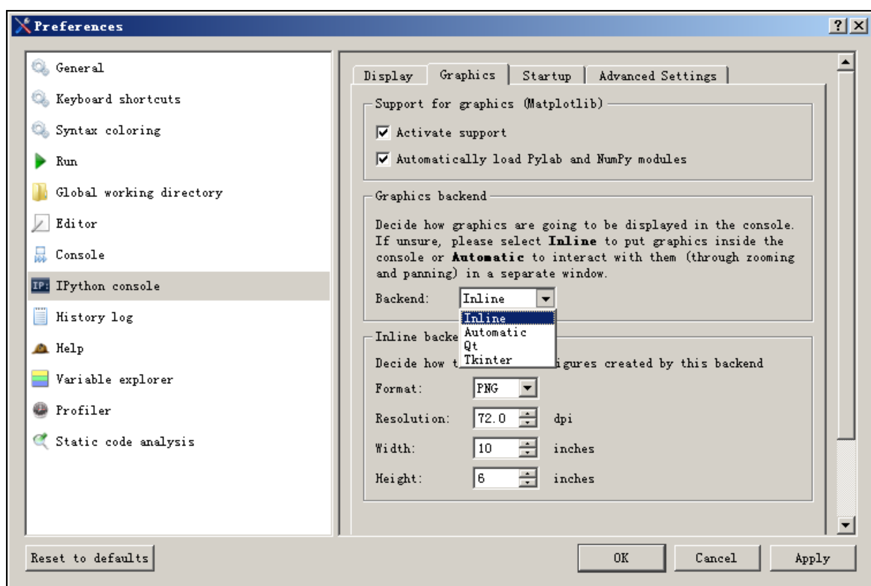


图 2-7　调整 Spyder 编辑器图像显示尺寸

4. 重剑无锋

对于量化分析的开发平台，笔者的一贯主张就是，直接使用 zwPython 内置的 Spyder 开发平台。

Spyder 的工作界面，经过多年一线量化分析实盘操作人员的反馈调整和设计优化，对于金融量化工作者而言，已经是一种非常理想的工作界面，也可以说是目前最好的工作界面，具体理由如下。

- Spyder（前身是 Pydee），是一个强大的交互式 Python 语言开发环境，提供高级的代码编辑、交互测试、调试等特性，支持包括 Windows、Linux 和 OS X 系统。
- Spyder 最早发布于 2009 年，经过多年的升级优化，目前已经非常成熟，最大程度上减少各种 Bug 对于实盘操作的干扰。
- Spyder 默认界面布局，如图 2-8 所示，类似 MATLAB，集中了代码编辑，项目管理，变量检查与图形查看等多种功能，这种界面布局也是金融工程、量化分析行业的标准工作界面。

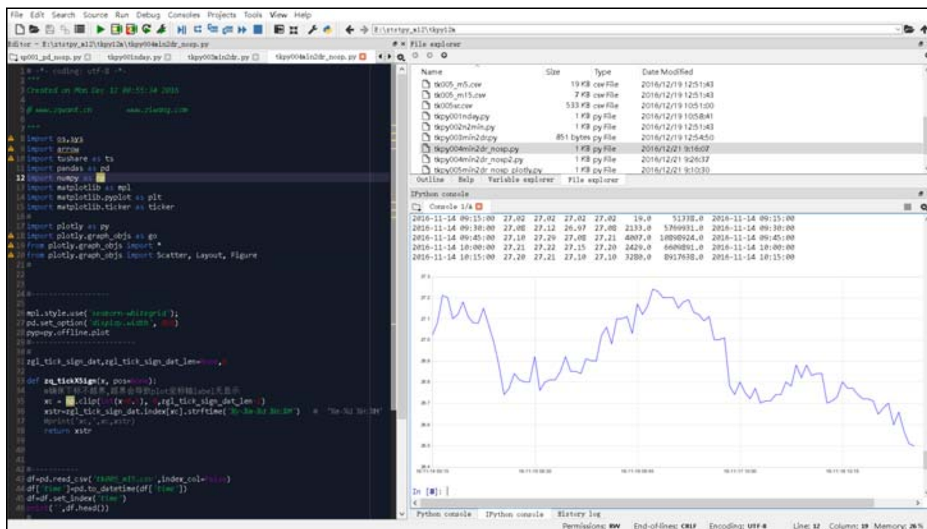


图 2-8　spyder 工作界面

GUI 用户界面，其实类似电脑的机箱，虽然华丽，但只是表层的东西，绝非核心因素。其实许多服务器采用的 Linux 操作系统，为了追求极致的性能，还在使用传统的纯文本界面，根本没有所谓的 GUI 用户接口。

笔者在设计 zwQuant 极宽量化软件时，无论是最初的 zwQuant，还是

zQuant-core（内核版），都强调简单实用，这些案例也可以说是“kiss 法则”（保持简单）在软件工程中的具体应用。

也许，大家觉得 Spyder 的界面过于朴素，这种朴素其实源自开源的历史与传承，大家可以看看一些著名开源项目的网站，网页都非常简单朴素，有些甚至还是互联网起步阶段的文本模式。

- http://www.apache.org：apach 开源项目网站。
- https://github.com：GITHUB 开源项目网站。
- http://www.lfd.uci.edu/~gohlke/Pythonlibs：LFD 二进制 Python 模块库。
- http://mirrors.163.com/：网易开源镜像网站。
- http://mirrors.sohu.com/：搜狐开源镜像网站。

幸运的是，如今很多成功的互联网企业，如谷歌、百度也继承了这种朴素简练的传统，搜索引擎的首页都是大片的空白，类似中国传统书法的留空，只有简简单单的搜索框。

在这种朴素的背后是一种“重剑无锋”的体现。

最后，笔者再一次强调，量化分析的核心是策略，而不是交易接口、自动下单，也不是用户界面，更不是软件开发平台。

目前，Python 语言已经是数据分析、量化分析的行业标准编程语言，大家无须争议。

在大家试图质疑这些问题的时候，请好好重新审视一下软件工程的名言：

“Don't Reinvent the Wheel”——不要重复发明轮子。

2.5 Python命令行模式

Python 命令行模式与普通的命令行模式不同，因为集成了 Python 的运行环境参数。

许多新用户都是直接使用 Win 或其他软件自带的 Dos 命令，进入 Dos 命令行，运行 pip 命令会出错。

正确的方法是，运行 Python 目录下的：WinPython Command Prompt.exe 程序，如图 2-9 所示。



图 2-9　WinPython Command Prompt.exe 程序

- Python 27 版本，py27\WinPython Command Prompt.exe。
- Python 35 版本，py35\WinPython Command Prompt.exe。

运行后，会自动进入 Python 对应的子目录。

- Python 27 版本的目录是：x:\zwPython\py27\Python-2.7.10.amd64\。
- Python 35 版本进行了优化，目录是：x:\zwPython\py35\scripts\。

2.6　Notebook模式

zwPython 内置的 Notebook 支持模式，目前已经是 Python 源码交流的常用模式，事实上，Notebook 已经是数据分析信息分享的 Web 标准模式。

Notebook 模式文件的后缀名是：.ipynb，类似 IE 的 MHT 网页打包格式，支持文字格式、排版、图像。

运行方法如下：

- 进入 Python 35 目录
- 单击运行 Jupyter Notebook.exe 程序

Jupyter Notebook.exe 程序类似单机的本地 Web 服务器软件。

如图 2-10 所示，程序运行后会自动调用默认浏览器，并访问默认网址：http://localhost:8888/tree。

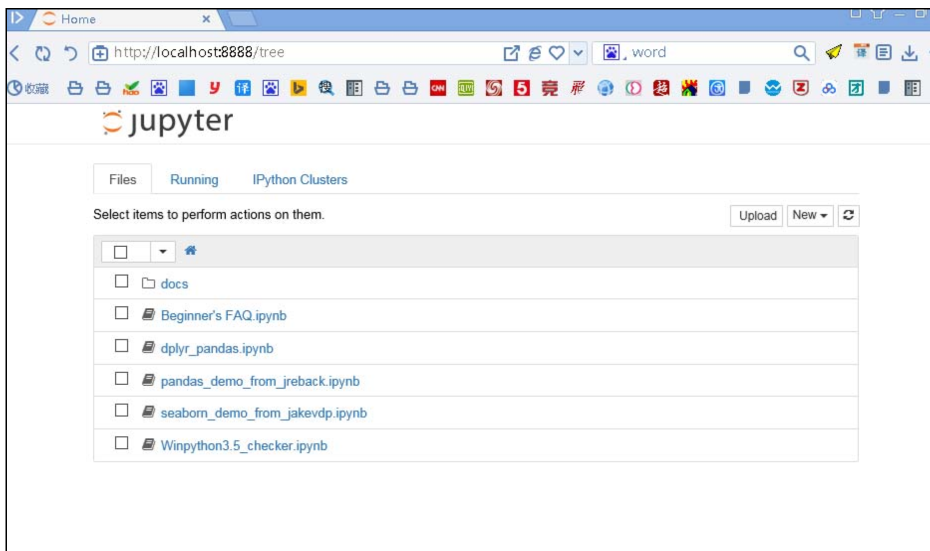


图 2-10　Notebook 模式

“.ipynb”格式文件使用方法如下。

- 运行 Jupyter Notebook.exe 程序，进入 Notebook 模式。
- 单击右上角的“Upload”按钮，或者用鼠标直接拖放“.ipynb”格式文件到浏览器窗口。
- 再单击文件名右侧的“Upload”按钮即可上传文件。
- 上传文件后，单击相应的文件名，即可看到相应的脚本内容，以及运行结果和图片。

具体效果如图 2-11 所示，但效果图会根据文件内容不同而有所不同。

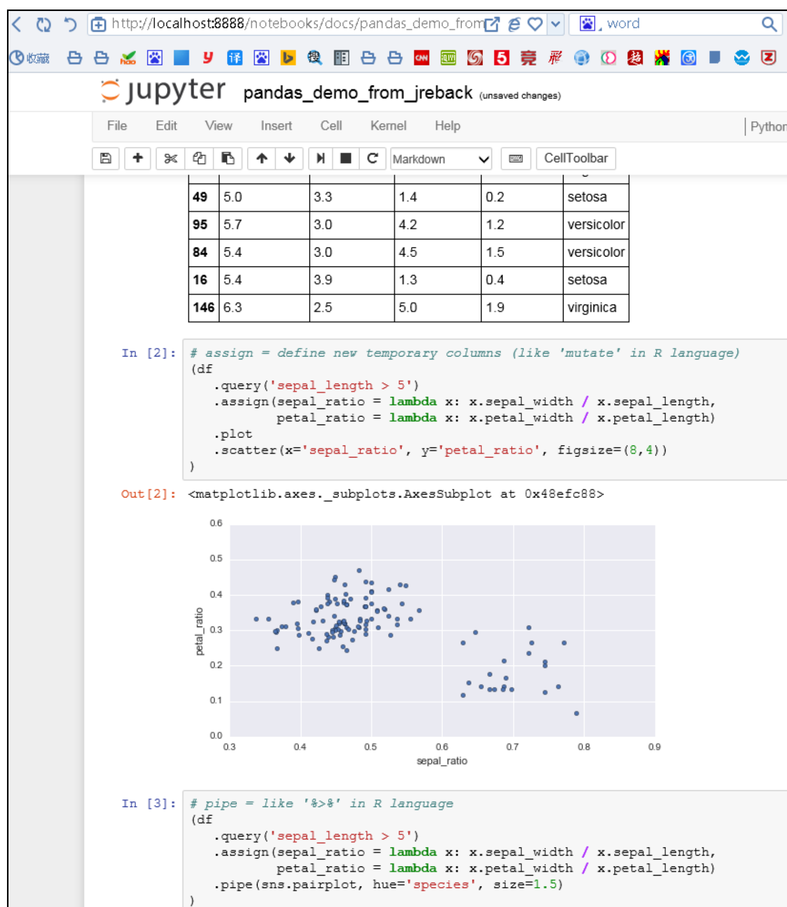


图 2-11　Notebook 运行模式

2.7　模块库控制面板

Python 的强大和方便除了体现在海量的内置模块外，还体现在绿色、灵活的模块库管理功能。

1. 模块库更新与增删

zwPython 的模块库管理直接使用 WinPython 的控制面板程序：WinPython Control Panel.exe 。

控制面板程序：WinPython Control Panel.exe，位于 Python 35 目录下，不同版本位置不同，不能混用，请大家注意。

运行后界面如图 2-12 所示。

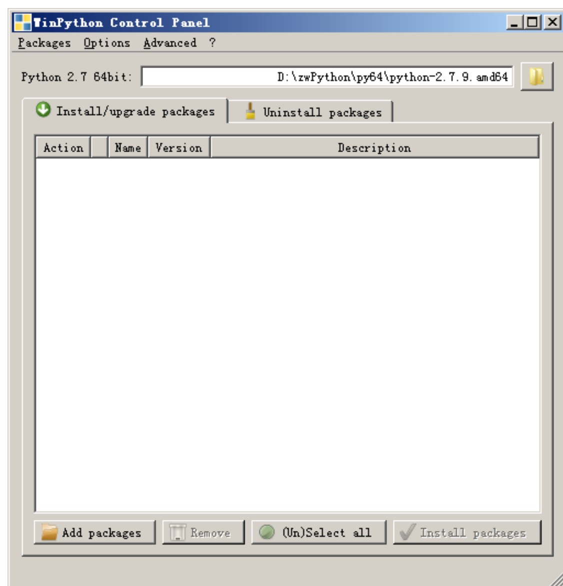


图 2-12　控制面板程序运行界面

2. zwPython模块库的安装流程

zwPython 模块库的安装流程如下。

- 把下载的 Python 模块库复制到任意一个目录。
- 单击右下角的“Add packages”按钮，从模块目录选择模块文件名即可完成模块库的添加。一次可选择添加多个模块库文件，如果模块库版本不对，则会出现提示对话框显示相关的出错模块名称；zwPython 系统是基于 64 位 Python 3.x 版本，因此下载模块，请选择对应的版本。
- 添加完毕后，单击右下方的“Install packages”即可完成模块库的安装。

需要注意的是：

- 模块安装完成后可以删除相关的模块文件，不影响程序使用。
- 多个模块安装时，每次最好不要超过 20 个，以免出错。

3. 模块库资源

zwPython 模块库资源，主要来自以下三个方面。

- 各大网络 Python 社区：主要是“.zip”、“.gz”格式。
- PyPI（Python Package Index）：Python 官方模块库，主要是“.zip”、“.gz"格式。
- LFD：加州欧文大学的非官方 Python 集成模块库，主要是“.exe”、“.whl”格式。
- Github：全球最大的程序资源网站，注意选择 Python 语言版本。

运行控制面板程序：WinPython Control Panel.exe 后，单击右下角的“Add packages”按钮，可以发现系统支持多种格式的模块库安装：“.zip”、“.gz"、“.exe”、“.whl”。

zwPython 在模块库安装方面的强大，主要体现在以下方面。

- 支持多种格式：除官方的“.zip”、“.gz”格式外，还支持 LFD 的“.exe”、“.whl”格式。
- 绿色安装：一次安装，随处运行，支持 U 主便携式开发。

Python 官方模块库 Pil 网址：https://pypi.Python.org/pypi。

Github 网址：https://github.org。

LFD 非官方模块资源网址：http://www.lfd.uci.edu/~gohlke/Pythonlibs/。（LFD 采用集成方式打包，特别适用于 Opencv、CUDA 等大型模块库安装）

LFD 全称是：Laboratory for Fluorescence Dynamics, University of California, Irvine.，动力学实验室，加利福尼亚大学/加州欧文大学。

加州欧文大学（简称为 UCI 或 UC Irvine，又常被译作加州大学欧文

分校）成立于 1965 年，是加州大学 10 个校区之一，位于美国加州。

4. 模块库维护更新

运行控制面板程序：WinPython Control Panel.exe，还提供了模块库的维护和升级功能，如图 2-13 所示，单击菜单：Options→Repair packages。

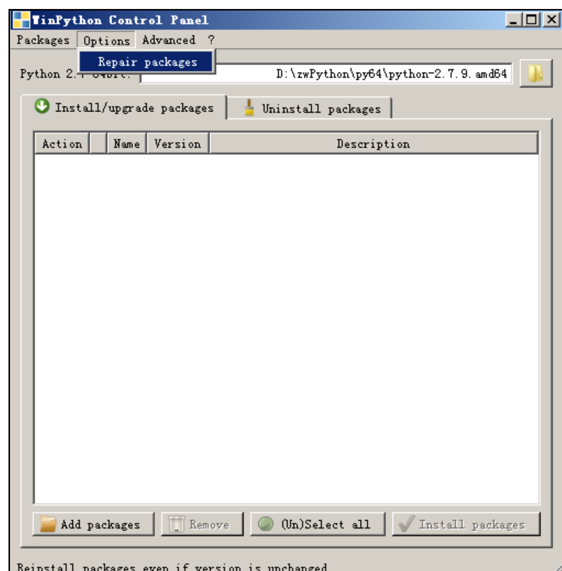


图 2-13 模块库维护

5. 系统关联

如图 2-14 所示，运行控制面板程序：WinPython Control Panel.exe，还提供系统关联功能（但通常无须采用关联模式）。

- 单击菜单：Advanced→Register...，即可将 zwPython 关联到 Windows 系统，关联后，可以直接在资源浏览器运行“.py”脚本文件，另外增加鼠标右键的“.py”脚本文件与“spyder”IDE 程序的关联编辑功能。
- 单击菜单：Advanced→Unregister...，即可解除关联。

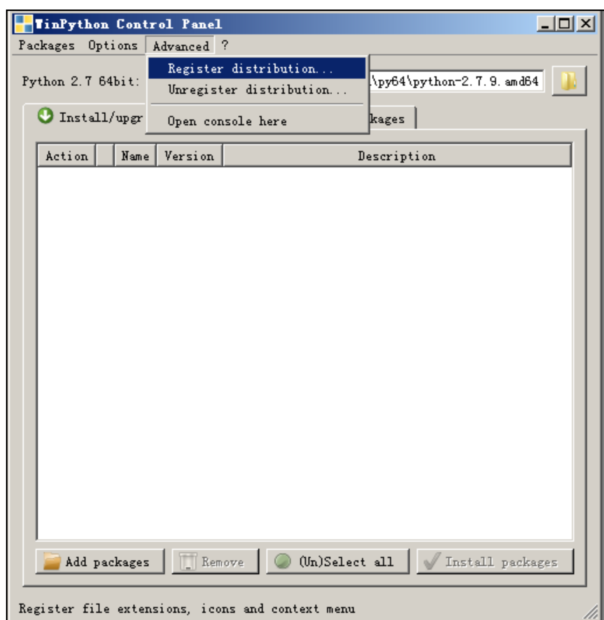


图 2-14　系统关联

2.8　使用pip更新模块库

有时，由于各种原因，使用控制台安装模块库会出现失败现象，或者需要批量更新模块库，这时，可以使用 pip 模块管理程序。

1. pip常用命令

（1）列出已安装的包

```
pip freeze or pip list
```

（2）导出 requirements.txt

```
pip freeze > <目录>/requirements.txt
```

（3）在线安装：安装包、模块库

```
pip install <包名>或 pip install -r requirements.txt
```

（4）指定版本

通过使用==、>=、<=、> <等符号来指定版本，不写则安装最新版。

requirements.txt 内容格式为：

```
APScheduler==2.1.2
Django==1.5.4
MySQL-Connector-Python==2.0.1
MySQL-Python==1.2.3
PIL==1.1.7
South==1.0.2
django-grappelli==2.6.3
django-pagination==1.0.7
```

（5）安装本地安装包

```
pip install <目录>/<文件名> 或 pip install --use-wheel --no-index --find-links=wheelhouse/ <包名>
<包名>前有空格
```

可简写为

```
pip install --no-index -f=<目录>/ <包名>
```

（6）卸载包

```
pip uninstall <包名> 或 pip uninstall -r requirements.txt
```

（7）升级包

```
pip install -U <包名>
```

（8）升级 pip

```
pip install -U pip
```

（9）显示包所在的目录

```
pip show -f <包名>
```

（10）搜索包

```
pip search <搜索关键字>
```

（11）查询可升级的包

```
pip list -o
```

（12）下载包而不安装

```
pip install <包名> -d <目录> 或 pip install -d <目录> -r requirements.txt
```

（13）安装 wheel 格式的模块库

```
pip wheel <包名>
```

（14）国内 pypi 镜像

```
pypi.v2ex.com/simple    V2EX
http://pypi.douban.com/simple  豆瓣
http://mirrors.aliyun.com/pypi/simple/ 阿里云（推荐使用）
http://pypi.mirrors.ustc.edu.cn/simple/    中国科学技术大学
https://pypi.tuna.tsinghua.edu.cn/simple      清华大学
http://pypi.hustunique.com/  华中理工大学
http://pypi.sdutlinux.org/  山东理工大学
http://pypi.mirrors.ustc.edu.cn/  中国科学技术大学
http://mirrors.sohu.com/Python/ 搜狐镜像
```

（15）指定镜像安装源

```
pip install <包名> -i http://pypi.v2ex.com/simple
```

其他更多有关 pip 的使用细节，大家可以自行搜索。

2. pip安装模版

为了方便大家使用 pip 安装新的模块库，zwPython 集成了一个 pip01.bat 批命令模版，位于不同版本对应的目录下。

pip01.bat 批命令内容如下：

```
rem tushare
pip install --upgrade tushare  -i https://pypi.tuna.tsinghua.
edu.cn/simple
```

其中，tushare 是示例的模块库名称，请大家自行改为需要安装更新的模块库名称。

这个 pip01.bat 批命令会自动更新指定的模块库，如果找不到对应的模块，则会重新安装。

因为 Python 官网速度很慢，所以，我们在 pip01.bat 批命令中使用了国内的镜像源，如果出现网络问题，大家可以根据前面介绍的 pypi 镜像站点或者自行搜索，更换对应的镜像网站即可。

3. pip参数解释

pip 部分参数及其解释如表 2-1 所示。

表 2-1　pip 部分参数及其解释

参　　数	参数解释
install	安装包
uninstall	卸载包
freeze	按着一定格式输出已安装包列表
list	列出已安装包
show	显示包详细信息
search	搜索包，类似于 yum 里的 search
wheel	按 requirements.创建模块包
zip	不推荐，Zip individual packages
unzip	不推荐，Unzip individual packages
bundle	不推荐，Create pybundles
download	下载模块

续表

参　　数	参数解释
hash	计算模块包的 Hash 哈希数值
help	当前帮助
-h, --help	显示帮助
-v, --verbose	更多的输出，最多可以使用 3 次
-V, --version	显示版本信息，然后退出
-q, --quiet	最少的输出
--log-file <path>	覆盖记录 verbose 错误日志，默认文件/root/.pip/pip.log
--log <path>	不覆盖记录 verbose 输出的日志
--retries <retries>	重试次数（默认 5 次）
--trusted-host <hostname>	可信任站点，不包括 https 站点
--timeout <sec>	连接超时时间 （默认 15 秒）
--cert <path>	证书
--cache-dir <dir>	cache 目录
--isolated	绝对模式，无视 Python 环境和用户设置
--upgrade	如果已安装就升级到最新版

4. pip-install参数选项

install 是最常用的 pip 参数，install 参数选项如表 2-2 所示。

表 2-2　install 参数选项及其含义

参　　数	含　　义
-c, --constraint <file>	约束，使用给定的约束限制版本文件，此选项可多次使用
-e, --editable <path/url>	可编辑的<路径/网址>在“开发模式”下安装一个项目
-r, --requirement <file>	要求<文件>，按给定要求文件安装模块

续表

参　数	含　义
-b, --build <dir>	建造模块包
-t, --target <dir>	目标<目录>
-d, --download <dir>	下载到<目录>
--src <dir>	SRC 目录查看编辑项目
-U, --upgrade	所有的包升级到最新
--force-reinstall	升级时强制重新安装，即使它们已经是最新的了
-I, --ignore-installed	忽略已经安装的模块包（与 Reinstalling 相反）
--no-deps	不安装依赖包
--install-option <options>	安装选项<选项>，使用 setup.py 的额外参数
--global-option <options>	全局选项
--egg	采用 eggs 模式安装，不用默认的“flat”模式
--root <dir>	根目录，安装使用的根目录
--prefix <dir>	安装前缀目录
--compile	编译 Py 文件为 Pyc 代码
--no-compile	不编译 Py 文件为 Pyc 代码
--no-use-wheel	不使用 Wheel 模块包
--no-binary <format_control>	不使用二进制模块包
--only-binary <format_control>	不使用源码模块包，只用二进制模块包
--pre	包括预处理和开发版本
--no-clean	不清除 Build 创建目录
--require-hashes	使用 Hash 验证

3

第 3 章

Python 入门案例

考虑到很多刚入门的读者对于 Python 不甚了解，所以在此特意增加了一章 Python 入门案例套餐，通过几个简单的 Python 入门程序，帮助大家尽快掌握 Python 语言，熟悉开发环境。

本书采用的是逆向式 Python 语言教学模式，先通过解压即用的 zwPython 开发平台和入门案例套餐，让大家对于 Python 语言有个基本的感性认识，再开始介绍 Python 语言的基本语法。

3.1 案例3-1：第一次编程“ hello,ziwang ”

软件安装完毕，我们就可以开始编写、运行 Python 脚本程序了。

- 单击工具栏的“读取”按钮，打开“py_demo\”目录下的 py301.py 脚本文件。
- 单击工具栏中部的绿色“▸”运行按钮。

程序很简单，只有一行代码：

```
print("hello,ziwang.com")
```

如图 3-1 所示，运行后，在右下角的输出窗口可以看到“hello,ziwang.com”的字样，表示运行成功。注意，输出面板是 IPython console。

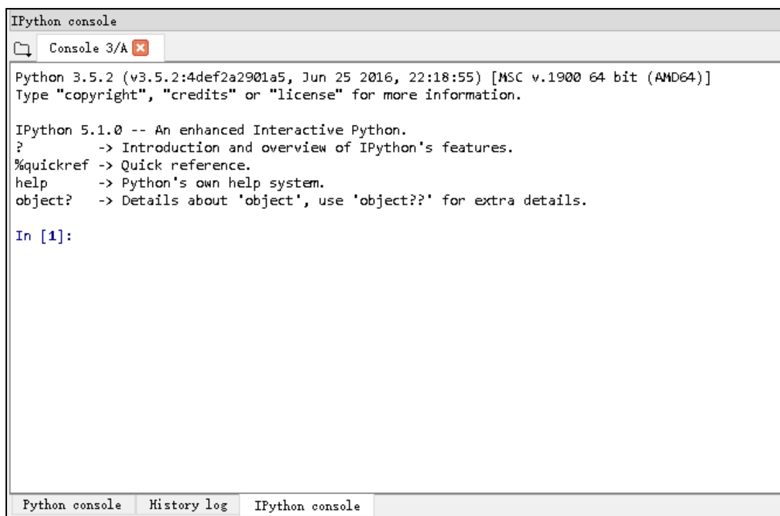


图 3-1 输出面板

大家可以自己修改引号里面的文字，看看输出效果，注意，此处必须是英文字符和标点，中文字符我们后面再说。

1. 简单调试

下面，我们学习最简单的调试，如图 3-2 所示，去掉代码左边的引号，再按“▸”运行按钮。

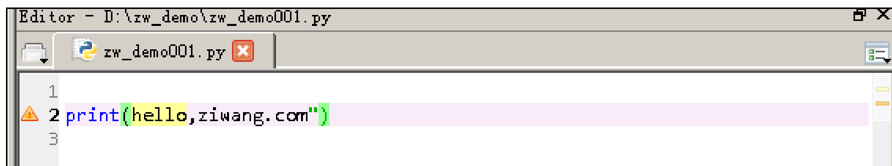


图 3-2 修改代码

右下角的输出窗口如图 3-3 所示。

```
IPython console
Console 3/A

Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('E:/zwPython/zc_dmo/zc301.py', wdir='E:/zwPython/zc_dmo')
  File "E:/zwPython/zc_dmo/zc301.py", line 2
    print(hello,ziwang.com")
                           ^
SyntaxError: EOL while scanning string literal


In [2]:

Python console | History log | IPython console
```

图 3-3　输出窗口

图 3-3 显示输出有错误，注意这行文字：

```
File "e:/zwPython/py_demo/zc201.py", line 2
```

其中的“line 2”表示出错的代码位于第二行。

出错信息：

```
SyntaxError: EOL while scanning string literal
```

表示是字符串应用错误，我们加上引号即可。

2. 控制台复位

有时，由于脚本代码或者其他原因，可能引发严重错误，比如系统运行时出现死循环或崩溃问题。

如图 3-4 所示，单击 IDE 右侧中部的“Restart”下拉菜单和按钮，控制台重新复位即可。

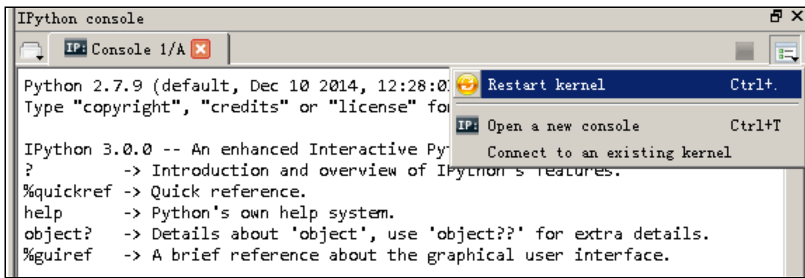


图 3-4　系统复位

3.2　案例3-2：增强版“hello,zwiang”

下面我们运行一个增强版的“hello,ziwang”。

- 单击工具栏的“读取”按钮，打开“py_demo\”目录下的 py302.py 脚本文件。
- 单击工具栏中部的绿色“▸”运行按钮。

案例 3-2.py 脚本文件很简单，核心程序才十几行，不过功能非常强大，除输出文字“hello”等信息外，还提供中文输出，以及检测系统多个重量级模块（比如 Opencv、Plotly、Pygame、Pandas 等）是否安装成功和版本是多少。

```
# -*- coding: utf-8 -*-

import sys,os,re
import cv2
import arrow,plotly

import pandas as pd
import tushare as ts
import pygame

print("hello,zwPython 2017")
```

```
    print("hello,TopQuant,TopFootball")
    print("极宽量化回溯系统，极宽足彩量化分析系统")
    print("")
    print("Python ver:",sys.version)
    print("")
    print("re ver:",re.__version__)
    print("arrow:",arrow.__version__)
    print("plotly:",plotly.__version__)
    print("")
    print("pandas ver:",pd.__version__)
    print("tushare ver:",ts.__version__)
    print("")
    print("pygame ver:",pygame.ver)
    print("opencv ver:",cv2.__version__)
```

如果案例 3-2 运行无误，输出面板，则结果如下（不同版本，细节略有差别）：

```
    hello,zwPython 2017
    hello,TopQuant,TopFootball
    极宽量化回溯系统，极宽足彩量化分析系统

    Python ver: 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55)
[MSC v.1900 64 bit (AMD64)]

    re ver: 2.2.1
    arrow: 0.10.0
    plotly: 2.0.0

    pandas ver: 0.19.1
    tushare ver: 0.6.2

    pygame ver: 1.9.2b1
    opencv ver: 3.1.0
```

案例 3-2 表明，即使是初学者，采用 Basic 的过程模式编写简单代码，也能完成很复杂的功能。

3.3 案例3-3：列举系统模块库清单

案例 3-3 文件名是 py_demo\py303mlst.py，代码很简单，如下所示：

```
# -*- coding: utf-8 -*-

import numpy as np
import scipy as sp
import pandas as pd
import pip

# =====================
x10=pip.get_installed_distributions();
df=pd.DataFrame();
df['name']=x10
print(df.head())

df.to_csv('tmp/m10.csv',index=False)
```

运行结果如下：

```
                         name
0               3to2 1.1.1
1      zope.interface 4.3.2
2  zipline-cn-databundle 0.4
3               zict 0.1.0
4               zarr 2.1.3
```

运行后，会在 tmp 目录生成一个名称为 m10.csv 的数据文件，文件里面是当前 zwPython 所安装的第三方模块库名称和版本号。在《zwPython 用户手册》中，《zwPy3.5 内置模块库列表》就是这样完成的。

案例 3-3 代码虽然简单，但调用了 Pandas（潘达思）数据分析模块，以及很少在程序中直接调用的 pip 模块库，来获取内置模块清单。

案例 3-3 是笔者在编程中实际使用的脚本程序，大家可以自行保存，用于日常查看自己的 Python 运行环境的模块库安装情况。

3.4　案例3-4：常用绘图风格

可视化计算是数据分析的重要组成部分，Python 可视化计算、数据分析和量化分析最重要的模块是：Matplotlib。

如今，虽然新一代绘图模块 Plotly 横空出世，且在互动性方面具有压倒性的优势，但由于历史原因，Pandas（潘达思）数据分析模块和很多第三方内置的绘图模块还是使用 Matplotlib，绘制一般的简单图形，还是使用 Matplotlib 比较方便。

老版本的 Matplotlib 默认的绘图风格与现代设计风格有些差距，因此，需要增加一些扩展风格包。2017 年 1 月，期待已久的 Matplotlib 2.0 终于发布，在风格方面有所优化。

通常，Matplotlib 的绘图风格只有几种：

```
    'bmh','dark_background','fivethirtyeight','ggplot','grayscale
','default'。
```

zwPython 将 Matplotlib 的绘图风格增加到 20 多种：

```
    'dark_background', 'seaborn-colorblind', 'classic', 'grayscale',
'seaborn-dark-palette','seaborn-ticks','fivethirtyeight','seaborn
-paper','seaborn-poster','seaborn-white','seaborn-muted','seaborn
-notebook','seaborn-dark','bmh','ggplot','seaborn-darkgrid','seab
orn-whitegrid','seaborn-bright','seaborn-pastel','seaborn-talk','
seaborn-deep'
```

以上风格，除 Matplotlib 调用外，也可直接应用到 Pandas 绘图语句当

中，需要注意的是，使用不同的版本、处于不同的 Python 运行环境时，以上具体参数在细节方面会有所差异。

案例 3-4 文件名为 py304dr.py，代码使用 Matplotlib 绘图指令进行绘图，全部代码如下：

```
    # -*- coding: utf-8 -*-

    import numpy as np
    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import pandas as pd

    def dr_xtyp(_dat):
        #xtyp=['bmh','dark_background','fivethirtyeight','ggplot',
'grayscale','default'];
        for xss in plt.style.available:
            plt.style.use(xss);print(xss)
            plt.plot(_dat['Open'])
            plt.plot(_dat['Close'])
            plt.plot(_dat['High'])
            plt.plot(_dat['Low'])
            fss="tmp\\stk001_"+xss+".png";plt.savefig(fss);
            plt.show()

    # =====================
    df = pd.read_csv('dat\\appl2014.csv',index_col=0,parse_dates=
[0],encoding='gbk')
    d30=df[:30];
    dr_xtyp(d30);
```

如图 3-5 所示，运行后会在 tmp 目录下生产一系列各种不同风格的图形，可以保存一下，作为日后编程绘图的参考。

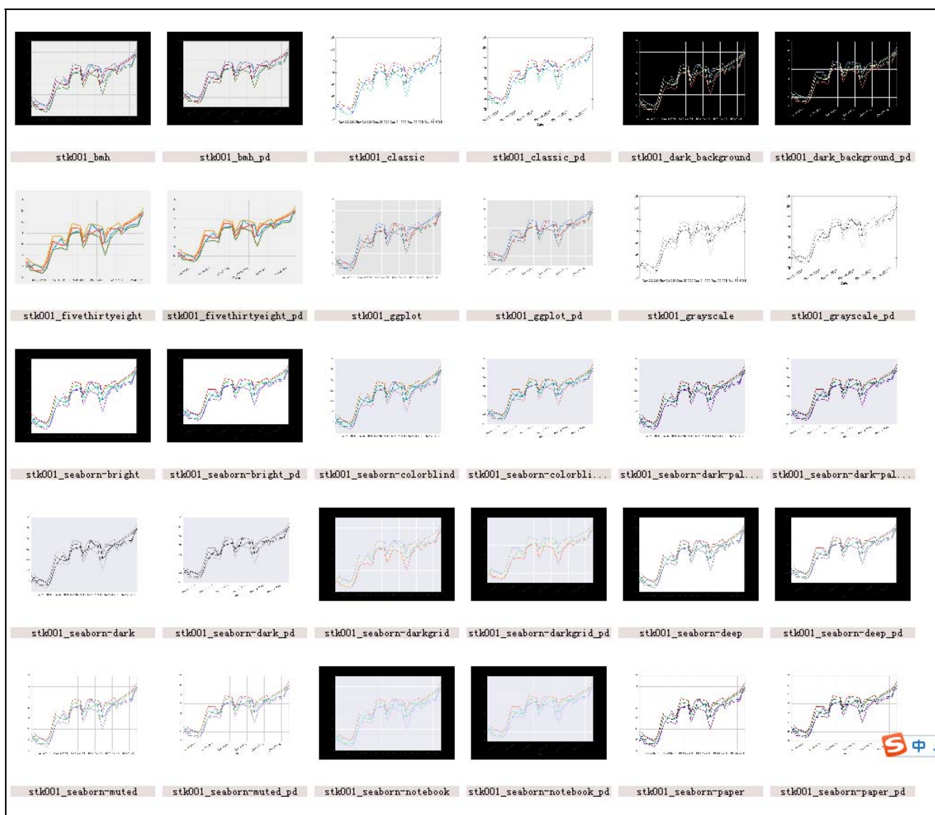


图 3-5　不同风格的输出图形

3.5　案例3-5：Pandas常用绘图风格

案例 3-5 文件名为 py305drpd.py，使用 Pandas（潘达思）数据分析模块内置的 plot 命令绘制图形，代码如下：

```
# -*- coding: utf-8 -*-


import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
```

```
    def dr_xtyp(_dat):
        #xtyp=['bmh','dark_background','fivethirtyeight','ggplot',
'grayscale','default'];
        for xss in plt.style.available:
            plt.style.use(xss);print(xss)
            _dat['Open'].plot();
            _dat['Close'].plot();
            _dat['High'].plot();
            _dat['Low'].plot();
            fss="tmp\\stk001_"+xss+"_pd.png";plt.savefig(fss);
            plt.show()

    # =======================
    df = pd.read_csv('dat\\appl2014.csv',index_col=0,parse_dates=
[0],encoding='gbk')

    d30=df[:30];
    dr_xtyp(d30);
```

案例 3-5 的运行结果与案例 3-4 类似，在此不再赘述。

比较案例 3-4 和案例 3-5 两段程序，只有以下语句不同：

```
plt.plot(_dat['Open'])
plt.plot(_dat['Close'])
plt.plot(_dat['High'])
plt.plot(_dat['Low'])
```

```
_dat['Open'].plot();
_dat['Close'].plot();
_dat['High'].plot();
_dat['Low'].plot();
```

运行后截图，除 Pandas 优化的细节外，两者风格大体相同。具体细节，请读者自己参看 tmp 目录下的相关图片。

Pandas（潘达思）数据分析模块是最重要的数据分析工具，大家必须了解其基本功能和常用的函数命令。

所以，笔者特意增加了案例 3-5，强调 Padnas（潘达思）数据分析软件

的使用，如果还有读者对于案例 3-5 有困惑，请补习一下 Padnas（潘达思）基础知识，再继续深入学习。

3.6　案例3-6：常用颜色表cors

下面我们再介绍一些常用的颜色组合，在极宽 zsys 模块中内置了多种常用的颜色组合，以下是 zsys 模块颜色组合设置的部分相关代码：

```
import matplotlib.colors
from matplotlib import cm

cors_brg=cm.brg(np.linspace(0,1,10))
cors_hot=cm.hot(np.linspace(0,1,10))
cors_hsv=cm.hsv(np.linspace(0,1,10))
cors_jet=cm.jet(np.linspace(0,1,10))
cors_prism=cm.prism(np.linspace(0,1,10))
cors_Dark2=cm.Dark2(np.linspace(0,1,10))
```

需要注意的是，以上的颜色组合代码没有使用常规的 rgb 参数设置模式，而是使用的 np.linspace 函数，从 Matplotlib 的 Corlormap 中提取 10 个颜色，组成一个系列的 10 种颜色，如果用户的参数超过 10 种，系统会自动循环使用每组的 10 个颜色。

大家也可以参考上面的 np.linspace 函数，并在案例 3-6 中使用 20、30、50 等其他参数，看看具体效果有什么不同。

案例 3-6 文件名为 py306_cors.py，说明如何绘制 Matplotlib 内置的颜色图谱，以及极宽量化模块内置的颜色组合。

颜色是图表设计的一个核心部分，事实上，现代图表与传统图表的一个重要差异就是颜色的组合，Matplotlib 2.0 升级版本，其中最重要的升级之一就是颜色体系更加适合现代的设计风格。

案例 3-6 内置了以下两个函数。

- dr_cormap()函数：绘制 Matplotlib 内置的颜色图谱，不同的 Python 环境，因 Matplotlib 版本和辅助模块的差异，具体数目会有所不同。
- dr_cors_sys()函数：绘制极宽量化模块内置的颜色组合，一共有 8 种。

dr_cormap()函数代码如下：

```
    def dr_cormap(fcor='dat/cormap.dat'):
        #font  =  FontProperties(fname=r"c:\windows\fonts\simsun.
ttc", size=14)
        clst=zt.f_lstRdTxt(fcor);
        ds=pd.Series(range(5,25));
        for xc,cor in enumerate(clst):
            css=cor[0]
            xss='cm.'+str(css)+'(np.linspace(0,1,10))'
            print(xc,'#',css,xss)
            cor2=eval(xss)
            #print(css,xss,cor2)
            ds.plot(kind='bar',rot=0,color=cor2)
            plt.savefig('tmp/cm_'+css+'.png')
```

程序运行后，会在 tmp 目录下生成数十个不同的颜色组合图形文件，大家可以选择自己喜欢的图形，参考极宽的颜色设置代码，添加自己喜欢的颜色组合。

案例 3-6 也是笔者在工作中实际使用的代码程序，大家可以浏览 dat/cormap.dat 文件，查看相关的颜色参数数据，保存好输出的图形文件，特别是自己喜欢的一些颜色效果，以供日后编程参考。

3.7 案例源码

为方便读者学习，我们特意把每一章的案例源码，统一集中在每章最后，以供读者参考。

本书代码，未来可能会根据版本的升级而不断优化，如果书中代码与

给出的下载路径中的源码有所不同，以最新下载的程序代码为准。

本章包括以下案例程序。

- 案例 3-1：第一次编程“hello,ziwang”，文件名是 py301.py。
- 案例 3-2：增强版“hello,zwiang”，文件名是 py302.py。
- 案例 3-3：列举系统模块库清单，文件名是 py303mlst.py。
- 案例 3-4：常用绘图风格，文件名是 py304dr.py。
- 案例 3-5：Pandas 常用绘图风格，文件名是 py305drpd.py。
- 案例 3-6：常用颜色表 cors，文件名是 py306cors.py。

案例 3-1：第一次编程“hello,ziwang”

案例 3-1：第一次编程“hello,ziwang”，文件名是 py301.py，源码如下。

```
print("hello,ziwang.com")
```

案例 3-2：增强版“hello,zwiang”

案例 3-2：增强版“hello,zwiang”，文件名是 py302.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com
'''


import sys,re
import cv2
import arrow,plotly

```

```
import pandas as pd
import tushare as ts
import pygame

print("hello,zwPython 2017")
print("hello,TopQuant,TopFootball")
print("极宽量化回溯系统，极宽足彩量化分析系统")
print("")
print("python ver:",sys.version)
print("")
print("re ver:",re.__version__)
print("arrow:",arrow.__version__)
print("plotly:",plotly.__version__)
print("")
print("pandas ver:",pd.__version__)
print("tushare ver:",ts.__version__)
print("")
print("pygame ver:",pygame.ver)
print("opencv ver:",cv2.__version__)
```

案例 3-3：列举系统模块库清单

案例 3-3：列举系统模块库清单，文件名是 py303mlst.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com
'''
import numpy as np
import scipy as sp
```

```
import pandas as pd
import pip

# ======================
x10=pip.get_installed_distributions();
df=pd.DataFrame();
df['name']=x10
print(df.head())

df.to_csv('tmp/m10.csv',index=False)
```

案例 3-4：常用绘图风格

案例 3-4：常用绘图风格，文件名是 py304dr.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com
'''


import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd

def dr_xtyp(_dat):
    #xtyp=['bmh','dark_background','fivethirtyeight','ggplot',
'grayscale','default'];
    for xss in plt.style.available:
        plt.style.use(xss);print(xss)
```

```
            plt.plot(_dat['Open'])
            plt.plot(_dat['Close'])
            plt.plot(_dat['High'])
            plt.plot(_dat['Low'])
            fss="tmp\\stk001_"+xss+".png";plt.savefig(fss);
            plt.show()

    # ======================
    df = pd.read_csv('dat\\appl2014.csv', index_col=0, parse_dates=
[0], encoding='gbk' )
    d30=df[:30];
    dr_xtyp(d30);
```

案例 3-5：Pandas常用绘图风格

案例 3-5：Pandas 常用绘图风格，文件名是 py305drpd.py，源码如下。

```
    #coding=utf-8
    '''
    Created on 2016.12.25
    Top Quant-极宽量化分析系统
    培训课件-配套教学 python 程序
    @ www.TopQuant.vip     www.ziwang.com
    '''
    import numpy as np
    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import pandas as pd

    def dr_xtyp(_dat):
        #xtyp=['bmh','dark_background','fivethirtyeight','ggplot',
'grayscale','default'];
        for xss in plt.style.available:
```

```
        plt.style.use(xss);print(xss)
        _dat['Open'].plot();
        _dat['Close'].plot();
        _dat['High'].plot();
        _dat['Low'].plot();
        fss="tmp\\stk001_"+xss+"_pd.png";plt.savefig(fss);
        plt.show()

# =======================
df = pd.read_csv('dat\\appl2014.csv', index_col=0, parse_dates=
[0], encoding='gbk')

d30=df[:30];
dr_xtyp(d30);
```

案例 3-6：常用颜色表cors

案例 3-6：常用颜色表 cors，文件名是 py306_cors.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com
'''

import pandas as pd
import numpy as np

import matplotlib as mpl
from matplotlib import pyplot as plt
from matplotlib import cm
```

```
import zsys
import ztools as zt
#

#----------------------
def dr_cormap(fcor='dat/cormap.dat'):
    #font  =  FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)
    clst=zt.f_lstRdTxt(fcor);
    ds=pd.Series(range(5,25));
    for xc,cor in enumerate(clst):
        css=cor[0]
        xss='cm.'+str(css)+'(np.linspace(0,1,10))'
        print(xc,'#',css,xss)
        cor2=eval(xss)
        #print(css,xss,cor2)
        ds.plot(kind='bar',rot=0,color=cor2)
        plt.savefig('tmp/cm_'+css+'.png')


def dr_cors_sys():
    ds=pd.Series(range(5,25));print(ds)
    ds.plot(kind='bar',rot=0,color=zsys.cors_prism)
    plt.savefig('tmp/prism.png')
    #
    ds.plot(kind='bar',rot=0,color=zsys.cors_brg)
    plt.savefig('tmp/brg.png')
    #
    ds.plot(kind='bar',rot=0,color=zsys.cors_Dark2)
    plt.savefig('tmp/dark2.png')
    #
    ds.plot(kind='bar',rot=0,color=zsys.cors_hsv)
    plt.savefig('tmp/hsv.png')
```

```
    #
    ds.plot(kind='bar',rot=0,color=zsys.cors_jet)
    plt.savefig('tmp/jet.png')
     #
    ds.plot(kind='bar',rot=0,color=zsys.cors_hsv)
    plt.savefig('tmp/hsv.png')
     #
    ds.plot(kind='bar',rot=0,color=zsys.cors_hot)
    plt.savefig('tmp/hot.png')
     #
    ds.plot(kind='bar',rot=0,color=zsys.cors_Vega10)
    plt.savefig('tmp/Vega10.png')
     #
    ds.plot(kind='bar',rot=0,color=zsys.cors_Vega20)
    plt.savefig('tmp/Vega20.png')



#-----------------------

dr_cormap()
dr_cors_sys()
#------------
#
print('\nok,
```

第 4 章 Python 基本语法

Python 是一种学习简单、功能强大的工业级编程语言，也是一种真正的终身编程语言，适合 8~80 岁的用户学习编程，是小学生和博士生通用的编程语言。

语法是编程语言中基础的基础，即使你是一位有经验的程序员，再看看这些 Python 编程语法，也会对于 Python 语言的细节有更多的认识。

4.1 数据类型

Python 有 5 种基本数据类型：

- Numbers（数字）
- String（字符串）
- List（列表）
- Tuple（元组）
- Dictionary（字典）

注意：

（1）Python 的数据类型和 C 语言的不同，其有复数形式，比如(-6+4j)和(5.3-7.6j)。

（2）Python 没有 char 单字符类型。

数字类型用于存储数值。

当给一个变量赋值时，Number 数据类型就会被创建：

```
x=1
y=911
```

Python 支持 4 种不同的数字类型：

- Int（有符号整型）
- Long（长整型[也可以代表八进制和十六进制]）
- Float（浮点型）
- Complex（复数）

Python 常用的计算符号有：

- +，加法。
- -，减法。
- *，乘法。
- /，除法。
- //，整除。
- %，取模，余数。
- **，乘方。

案例 4-1：基本运算

案例 4-1 文件名为 py401math.py，主要介绍 Python 数值的基本运算，核心代码如下：

```
#1
print('\n#1')
x=10
y=22
z=35
print('x,y,z,',x,y,z)

#2
print('\n#2')
a=x+y;print('a=x+y,',a)
b=x-y;print('b=x-y,',b)
c=z-x*y;print('c=z-x*y,',c)

#3
print('\n#3')
a=z/x;print('a=z/x,',a)
b=z//x;print('b=z//x,',b)
c=z%x;print('c=z%x,',c)

#4
print('\n#4')
a=x**2;print('a=x**2,',a)
b=x**3;print('b=x**3,',b)
```

对应的输出信息如下：

```
#1
x,y,z, 10 22 35

#2
a=x+y, 32
b=x-y, -12
c=z-x*y, -185

#3
```

```
a=z/x, 3.5
b=z//x, 3
c=z%x, 5

#4
a=x**2, 100
b=x**3, 1000
```

4.2　字符串

字符串 String 是由数字、字母、下画线组成的一串字符，一般采用单引号或者双引号形式：

```
str=’abcd’
str=”hello ziwang.com”
```

Python 语言的字符串类似传统语言的字符数组模式，也可以看作字符列表，有两种取值顺序：

- 从左到右索引默认 0 开始的，最大范围是字符串长度少 1。
- 从右到左索引默认-1 开始的，最大范围是字符串开头。

如果要实现从字符串中获取一段子字符串，使用变量 [头下标:尾下标]，就可以截取相应的字符串，其中下标从 0 开始算起，可以是正数或负数，下标可以为空，表示取到头或尾。

案例 4-2：字符串入门

案例 4-2 文件名为 py402str.py，主要介绍字符串的基本用法，核心代码如下：

```
dss='hello ziwang.com'
print('dss',dss)
```

```
#1
print('\n#1')
s2=dss[1:];print('s2,',s2)
s3=dss[1:3];print('s3,',s3)
s4=dss[:3];print('s4,',s4)

#2
print('\n#2')
s2=dss[-1];print('s2,',s2)
s3=dss[1:-2];print('s3,',s3)
dn=len(dss);print('dn,',dn)

#3
print('\n#3')
print('s2+s3,',s2+s3)
print('s3*2,',s3*2)
```

对应的输出信息如下：

```
dss hello ziwang.com

#1
s2, ello ziwang.com
s3, el
s4, hel

#2
s2, m
s3, ello ziwang.c
dn, 16

#3
s2+s3, mello ziwang.c
s3*2, ello ziwang.cello ziwang.c
```

在字符串运算中，加号（+）是字符串连接运算符，乘号（*）是重复操作。

案例 4-3：字符串常用方法

Python 语言的字符串其实是一种对象，内置了大量实用的字符串函数和方法，几乎包括了所有常用的 Python 字符串操作，如字符串的替换、删除、截取、复制、连接、比较、查找、分割等。

案例 4-3 文件名为 py403str2.py，主要介绍字符串内置函数和方法，所以，程序代码较长，我们分组进行说明。

程序代码	对应的输出信息
#1 dss='　hello ziwang.com,,' print('\n#1,去空格及特殊符号') s2=dss.strip().lstrip().rstrip(',') print('s2,',s2)	#1,去空格及特殊符号 s2, hello ziwang.com
#2 print('\n#2,字符串连接') s2=dss.join(['a','.','c']) print('s2,',s2) s3='s3' s3+='xx' print('s2,',s3)	#2,字符串连接 s2, a　hello ziwang.com,,.　hello ziwang.com,,c s2, s3xx
#3 print('\n#3,查找字符') css='abc1c2c3' pi=css.find('c') print('pi,',pi)	#3,查找字符 pi, 2
#4,字符串比较 s1>s2 False s1==s2 False s1<s2 True	#4,字符串比较 s1>s2 False s1==s2 False s1<s2 True

续表

程序代码	对应的输出信息
#5 print('\n#5,字符串长度') s1,s2='abc','c123' print('len(s1),',len(s1)) print('len(s2),',len(s2))	#5,字符串长度 len(s1), 3 len(s2), 4
#6 print('\n#6,大小写转换') s1,s2='abc','ABC123efg' print('大写，s1.upper(),',s1.upper()) print('小写，s2.lower(),',s2.lower()) print('大小写互换 ,s2.swapcase(),',s2.swapcase()) print('首字母大写 ,s1.capitalize(),',s1.capitalize())	#6,大小写转换 大写，s1.upper(), ABC 小写，s2.lower(), abc123efg 大小写互换 ,s2.swapcase(), abc123EFG 首字母大写 ,s1.capitalize(), Abc
#7 print('\n#7,分割字符串') s2=' hello, ziwang,com,,' print('s2.split,',s2.split(','))	#7,分割字符串 s2.split, [' hello', ' ziwang', 'com', '', '']

4.3 List列表

List 列表，使用[ ]标识，是 Python 中最常用的数据类型，List 列表类似传统语言中的数组，但更加灵活强大。

List 列表支持字符、数字、字符串，甚至可以包含列表（所谓嵌套）。

列表中的每一个元素都分配一个数字，即它的位置或索引，第一个索引是 0，第二个索引是 1，依此类推。从右到左索引默认从−1 开始，下标可以为空，表示取到头或尾。

案例 4-4：列表操作

案例 4-4 文件名为 py404list.py，主要介绍列表的基本用法，核心代码如下：

```
#1
print('\n#1')
zlst=['hello','ziwang','.','com']
vlst=['Top','Quant','.','vip']
print('zlst,',zlst)
print('vlst,',vlst)

#2
print('\n#2')
s2=zlst[1:];print('s2,',s2)
s3=zlst[1:3];print('s3,',s3)
s4=vlst[:3];print('s4,',s4)

#3
print('\n#3')
print('s2+s3,',s2+s3)
print('s3*2,',s3*2)
```

对应的输出信息如下：

```
#1
zlst, ['hello', 'ziwang', '.', 'com']
vlst, ['Top', 'Quant', '.', 'vip']

#2
s2, ['ziwang', '.', 'com']
s3, ['ziwang', '.']
s4, ['Top', 'Quant', '.']

#3
```

```
s2+s3, ['ziwang', '.', 'com', 'ziwang', '.']
s3*2, ['ziwang', '.', 'ziwang', '.']
```

加号（+）是列表连接运算符，乘号（*）是重复操作。

列表操作常用函数和方法

（1）List 列表操作包含以下函数。

- cmp(list1, list2)：比较两个列表的元素 。
- len(list)：列表元素个数。
- max(list)：返回列表元素最大值。
- min(list)：返回列表元素最小值。
- list(seq)：将元组转换为列表。

（2）List 列表操作包含以下方法。

- list.append(obj)：在列表末尾添加新的对象。
- list.count(obj)：统计某个元素在列表中出现的次数。
- list.extend(seq)：在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）。
- list.index(obj)：从列表中找出某个值第一个匹配项的索引位置。
- list.insert(index, obj)：将对象插入列表。
- list.pop(obj=list[-1])：移除列表中的一个元素（默认最后一个元素），并且返回该元素的值。
- list.remove(obj)：移除列表中某个值的第一个匹配项。
- list.reverse()：反向列表中元素。
- list.sort([func])：对原列表进行排序。

4.4 Tuple元组

Tuple 元组用“()”标识，是 List 列表数据格式的简化版本，不能二次

赋值，类似只读列表。

元组用“()”标识，内部元素用逗号隔开。但是元组不能二次赋值，类似只读列表。

案例 4-5：元组操作

案例 4-5 文件名为 py405tuple.py，主要介绍元组的基本用法，核心代码如下：

```
#1
print('\n#1')
zlst=('hello','ziwang','.','com')
vlst=('Top','Quant','.','vip')
print('zlst,',zlst)
print('vlst,',vlst)

#2
print('\n#2')
s2=zlst[1:];print('s2,',s2)
s3=zlst[1:3];print('s3,',s3)
s4=vlst[:3];print('s4,',s4)

#3
print('\n#3')
print('s2+s3,',s2+s3)
print('s3*2,',s3*2)
```

对应的输出信息如下：

```
#1
zlst, ('hello', 'ziwang', '.', 'com')
vlst, ('Top', 'Quant', '.', 'vip')
```

```
#2
s2, ('ziwang', '.', 'com')
s3, ('ziwang', '.')
s4, ('Top', 'Quant', '.')

#3
s2+s3, ('ziwang', '.', 'com', 'ziwang', '.')
s3*2, ('ziwang', '.', 'ziwang', '.')
```

4.5 Dictionary字典

Dictionary 字典，用“{ }”标识，由索引（key）和它对应的值（value）组成，是除了 List 列表以外，Python 中最灵活的内置数据结构类型，类似其他语言的 k-v 数据类型。列表是有序的对象结合，字典是无序的对象集合。

Python 字典是一种可变容器模型，且可存储任意类型对象，如字符串、数字、元组等其他容器模型。

字典由键和对应值成对组成，也被称作关联数组或哈希表，字典中的元素通过 key 关键词来存取，而不是通过偏移存取。

案例 4-6：字典操作

案例 4-6 文件名为 py406dict.py，主要介绍字典的基本用法，核心代码如下：

```
#1
print('\n#1')
zdict={}
zdict['w1']='hello'
zdict['w2']='ziwang.com'
```

```
    print('zdict,',zdict)



    #2
    print('\n#2')
    vdict={'url1':'TopQuant.vip'
          ,'url2':'www.TopQuant.vip'
          ,'url3':'ziwang.com'}
    print('vdict,',vdict)



    #3
    print('\n#3')
    s2=zdict['w1'];print('s2,',s2)
    s3=vdict['url2'];print('s3,',s3)
```

对应的输出信息如下：

```
    #1
    zdict, {'w2': 'ziwang.com', 'w1': 'hello'}

    #2
    vdict, {'url3': 'ziwang.com', 'url2': 'www.TopQuant.vip', 'url1':
'TopQuant.vip'}

    #3
    s2, hello
    s3, www.TopQuant.vip
```

字典内置函数和方法

（1）Python 字典包含以下内置函数。

- cmp(dict1, dict2)：比较两个字典元素。
- len(dict)：计算字典的元素个数，即键的总数。

- str(dict)：输出字典可打印的字符串标识。
- type(variable)：返回输入的变量类型，如果变量是字典就返回字典类型。

（2）Python 字典包含以下内置方法。

- radiansdict.clear()：删除字典内所有元素。
- radiansdict.copy()：返回一个字典的浅复制。
- radiansdict.fromkeys()：创建一个新字典，以序列 seq 中的元素做字典的键，val 为字典所有键对应的初始值。
- radiansdict.get(key, default=None)：返回指定键的值，如果值不在字典中，则返回 Default 值。
- radiansdict.has_key(key)：如果键在字典 Dictionary 里，则返回 True，否则返回 False。
- radiansdict.items()：以列表返回可遍历的（键，值）元组数组。
- radiansdict.keys()：以列表返回一个字典所有的键。
- radiansdict.setdefault(key, default=None)：和 get()类似，但如果键已经不存在于字典中，将会添加键并将值设为 Default。
- radiansdict.update(dict2)：把字典 Dictionary 2 的键/值对更新到 Dictionary 里。
- radiansdict.values()：以列表返回字典中的所有值。

4.6 数据类型转换

有时候，需要对数据内置的类型进行转换，只要将数据类型作为函数名即可。

以下几个内置的函数可以执行数据类型之间的转换，这些函数返回一个新的对象，表示转换的值。

常用的类型转换函数有以下几种。

- int(x [,base])：将 x 转换为一个整数。
- long(x [,base])：将 x 转换为一个长整数。
- float(x)：将 x 转换到一个浮点数。
- complex(real)：创建一个复数。
- str(x)：将对象 x 转换为字符串。
- repr(x)：将对象 x 转换为表达式字符串。
- eval(str)：用来计算在字符串中的有效 Python 表达式，并返回一个对象。
- tuple(s)：将序列 s 转换为一个元组。
- list(s)：将序列 s 转换为一个列表。
- chr(x)：将一个整数转换为一个字符。
- unichr(x)：将一个整数转换为 Unicode 字符。
- ord(x)：将一个字符转换为它的整数值。
- hex(x)：将一个整数转换为一个十六进制字符串。
- oct(x)：将一个整数转换为一个八进制字符串。

案例 4-7：控制语句

Python 语言的控制命令和其他编程语言类似，常用的有 if...else、while、for 等语句。

案例 4-7 文件名为 py407ctrl.py，介绍 Python 常用的控制命令，由于代码较长，下面进行分组说明。

第 1 组代码，说明 if...else 语句：

```
#1
print('\n1,if')
x,y,z=10,20,5
if x>y:
   print('x>y')
```

```
else:
   print('x<y')
```

对应的输出信息：

```
1,if
x<y
```

第 2 组代码，说明 elif 语句：

```
#2
print('\n#2,elif')
x,y,z=10,20,5
if x>y:
    print('x>y')
elif x>z:
   print('x>z')
```

对应的输出信息：

```
#2,elif
x>z
```

第 3 组代码，说明 while 循环语句：

```
#3
print('\n#3,whiel')
x=3
while x>0:
    print(x)
    x-=1
```

对应的输出信息：

```
#3,whiel
3
2
1
```

第 4 组代码，说明 for 循环语句的第一种用法：

```
#4
print('\n#4,for')
xlst=['1','b','xxx']
for x in xlst:
    print(x)
```

对应的输出信息：

```
#4,for
1
b
```

第 5 组代码，说明 for 循环语句的第二种用法：

```
#5
print('\n#5,for')
for x in range(3):
    print(x)
```

对应的输出信息：

```
#5,for
0
1
2
```

在 Python 控制语句中，需要注意 for 循环语句，其采用的迭代模式与传统编程语言差异较大，其他的控制语句用法都差不多。

案例 4-8：函数定义

Python 语言没有子程序，只有自定义函数，目的是方便我们重复使用相同的一段程序。将常用的代码块定义为一个函数，以备在以后你想实现

相同的操作时，只要调用函数名就可以了，而不需要重复输入所有的语句。函数的定义使用 def 命令。

案例 4-8 文件名为 py408fun.py，说明自定义函数的使用，核心代码如下：

```
def f01(a,b,c):
    print('a,b,c,',a,b,c)
    a2,b2,c2,=a+c,b*2,c*2
    #
    return a2,b2,c2
#

#1
print('\n#1')
x,y,z=f01(1,2,3)
print('x,y,z,',x,y,z)

#2
print('\n#2')
x,y,z=f01(x,y,z)
print('x,y,z,',x,y,z)
```

运行结果如下：

```
#1
a,b,c, 1 2 3
x,y,z, 4 4 6

#2
a,b,c, 4 4 6
x,y,z, 10 8 12
```

以上代码和输出信息需要注意的是：

- a、b、c 的输出信息是由自定义函数 f01 完成的。
- 调用 f01 函数时，变量 x、y、z 即是输入参数，也是输出变量。
- Python 函数支持多个返回数据。

4.7　案例源码

本章包括以下案例程序。

- 案例 4-1：基本运算，文件名是 py401math.py。
- 案例 4-2：字符串入门，文件名是 py402str.py。
- 案例 4-3：字符串常用方法，文件名是 py403str2.py。
- 案例 4-4：列表操作，文件名是 py404list.py。
- 案例 4-5：元组操作，文件名是 py405tuple.py。
- 案例 4-6：字典操作，文件名是 py406dict.py。
- 案例 4-7：控制语句，文件名是 py407ctrl.py。
- 案例 4-8：函数定义，文件名是 py408fun.py。

案例 4-1：基本运算

案例 4-1：基本运算，文件名是 py401math.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com
'''


#------------------
#1
print('\n#1')
x=10
y=22
z=35
print('x,y,z,',x,y,z)
```

```
#2
print('\n#2')
a=x+y;print('a=x+y,',a)
b=x-y;print('b=x-y,',b)
c=z-x*y;print('c=z-x*y,',c)

#3
print('\n#3')
a=z/x;print('a=z/x,',a)
b=z//x;print('b=z//x,',b)
c=z%x;print('c=z%x,',c)

#4
print('\n#4')
a=x**2;print('a=x**2,',a)
b=x**3;print('b=x**3,',b)
```

案例 4-2：字符串入门

案例 4-2：字符串入门，文件名是 py402str.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com
'''


#------------------
dss='hello ziwang.com'
print('dss',dss)
```

```
#1
print('\n#1')
s2=dss[1:];print('s2,',s2)
s3=dss[1:3];print('s3,',s3)
s4=dss[:3];print('s4,',s4)

#2
print('\n#2')
s2=dss[-1];print('s2,',s2)
s3=dss[1:-2];print('s3,',s3)
dn=len(dss);print('dn,',dn)

#3
print('\n#3')
print('s2+s3,',s2+s3)
print('s3*2,',s3*2)
```

案例 4-3：字符串常用方法

案例 4-3：字符串常用方法，文件名是 py403str2.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com
'''


#-----------------


#1
```

```
dss='  hello ziwang.com,,'
print('\n#1,去空格及特殊符号')
s2=dss.strip().lstrip().rstrip(',')
print('s2,',s2)

#2
print('\n#2,字符串连接')
s2=dss.join(['a','.','c'])
print('s2,',s2)
s3='s3'
s3+='xx'
print('s2,',s3)

#3
print('\n#3,查找字符')
css='abc1c2c3'
pi=css.find('c')
print('pi,',pi)

#4
print('\n#4,字符串比较')
s1='abc'
s2='c123'
print('s1>s2',s1>s2)
print('s1==s2',s1==s2)
print('s1<s2',s1<s2)

#5
print('\n#5,字符串长度')
s1,s2='abc','c123'
print('len(s1),',len(s1))
print('len(s2),',len(s2))

#6
```

```
print('\n#6,大小写转换')
s1,s2='abc','ABC123efg'
print('大写, s1.upper(),',s1.upper())
print('小写, s2.lower(),',s2.lower())
print('大小写互换 ,s2.swapcase(),',s2.swapcase())
print('首字母大写 ,s1.capitalize(),',s1.capitalize())

#7
print('\n#7,分割字符串')
s2='  hello, ziwang,com,,'
print('s2.split,',s2.split(','))
```

案例 4-4：列表操作

案例 4-4：列表操作，文件名是 py404list.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com
'''


#------------------
#1
print('\n#1')
zlst=['hello','ziwang','.','com']
vlst=['Top','Quant','.','vip']
print('zlst,',zlst)
print('vlst,',vlst)

#2
```

```
print('\n#2')
s2=zlst[1:];print('s2,',s2)
s3=zlst[1:3];print('s3,',s3)
s4=vlst[:3];print('s4,',s4)

#3
print('\n#3')
print('s2+s3,',s2+s3)
print('s3*2,',s3*2)
```

案例 4-5：元组操作

案例 4-5：元组操作，文件名是 py405tuple.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com
'''

#-----------------

#1
print('\n#1')
zdict={}
zdict['w1']='hello'
zdict['w2']='ziwang.com'
print('zdict,',zdict)

#2
print('\n#2')
```

```
vdict={'url1':'TopQuant.vip'
      ,'url2':'www.TopQuant.vip'
      ,'url3':'ziwang.com'}
print('vdict,',vdict)

#3
print('\n#3')
s2=zdict['w1'];print('s2,',s2)
s3=vdict['url2'];print('s3,',s3)
```

案例 4-6：字典操作

案例 4-6：字典操作，文件名是 py406dict.py，源码如下。

```
案例 4-6 字典操作，文件名 py406dict.py，源码如下。
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com
'''

#------------------
#1
print('\n#1')
zlst=('hello','ziwang','.','com')
vlst=('Top','Quant','.','vip')
print('zlst,',zlst)
print('vlst,',vlst)

#2
print('\n#2')
```

```
s2=zlst[1:];print('s2,',s2)
s3=zlst[1:3];print('s3,',s3)
s4=vlst[:3];print('s4,',s4)

#3
print('\n#3')
print('s2+s3,',s2+s3)
print('s3*2,',s3*2)
```

案例 4-7：控制语句

案例 4-7：控制语句，文件名是 py407ctrl.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com
'''

#------------------
#1
print('\n1,if')
x,y,z=10,20,5
if x>y:
    print('x>y')
else:
   print('x<y')



#2
print('\n#2,elif')
```

```
x,y,z=10,20,5
if x>y:
    print('x>y')
elif x>z:
   print('x>z')

#3
print('\n#3,whiel')
x=3
while x>0:
    print(x)
    x-=1

#4
print('\n#4,for')
xlst=['1','b','xxx']
for x in xlst:
    print(x)

#5
print('\n#5,for')
for x in range(3):
    print(x)
```

案例 4-8：函数定义

案例 4-8：函数定义，文件名是 py408fun.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
Top Quant-极宽量化分析系统
培训课件-配套教学 python 程序
```

```
@ www.TopQuant.vip     www.ziwang.com
'''


#------------------
def f01(a,b,c):
    print('a,b,c,',a,b,c)
    a2,b2,c2,=a+c,b*2,c*2
    #
    return a2,b2,c2
#

#1
print('\n#1')
x,y,z=f01(1,2,3)
print('x,y,z,',x,y,z)

#2
print('\n#2')
x,y,z=f01(x,y,z)
print('x,y,z,',x,y,z)
```

5 第 5 章 Python 人工智能入门与实践

5.1 从忘却开始

广大初学者面对人工智能、机器学习这些高大上的概念，一方面，迫切希望能够掌握相关的知识；另一方面，面对各种层出不穷的概念，往往会眼花缭乱、无所适从。

目前 Python 已经是人工智能、机器学习的行业标准语言，TensorFlow、Torch 于 2015 年、2016 年先后开放了 Python 语言接口。

Sklearn 是 Python 语言最重要的人工智能模块库，目前已经收入 Scikit 套件，所以又称为 Scikit-Learn，不过通常还是简称为 Sklearn。

如图 5-1 所示是 Sklearn 官方网站的人工智能、机器学习知识图谱。



图 5-1　人工智能、机器学习知识图谱

不知道各位读者对此图的感觉是什么，反正，笔者看到这幅图的第一反应是吓呆了，也明白了为什么这么多初学者对于人工智能望而生畏。

人工智能、机器学习、数据分析是笔者研究了多年的课题与领域，20 年前笔者最早的论文标题就是《人工智能与中文字型设计》，该论文还在暴雪（你没看错，就是做魔兽的暴雪）、宝洁等国际 500 强的知识产权案件中，被作为技术文献引用。

既然笔者的论文能够用于非常严肃的司法领域，那么在人工智能领域，笔者也可以算是根正苗红的学者了。

与普通理论学者不同的是，笔者始终认为自己是一名软件工程师或者程序员，也就是网络上常说的“码农”。

对于程序员而言：Talk is cheap, show me the code！

再多的理论也比不上几件成功的软件作品，笔者虽然谈不上有很多成功的软件作品，但也写过不少专业的程序，例如原生的 OCR 识别程序和英汉翻译程序。

所谓原生程序就是直接采用 C 语言、Delphi 语言的标准函数库直接编写应用软件，没有采用任何第三方 AI 架构库，如 Sklearn、TensorFlow 等 AI 模块库。做过底层系统的程序员，以及阅读过 Linux、安卓系统、TensorFlow 系统源码的程序员，会明白其中的难度。

这种基于原生的开发，最大的好处是，无论采用何种编程语言、何种理论算法，最终的底层代码结构都差不多，就像电脑里面的 CPU，算得再快也不过是个“加法器”。

笔者曾经在博客《文科生、易经与大数据》中说过：

殊途同归，什么东西到了极致，其根源都是相通的。

易经是纯文科的了，“字王小数据理论”“黑天鹅算法”的灵感就是来自易经、阴阳、八卦……

以量化投资为例，在进行大数据分析时，我们发现，所有的分析，抛开表象，到了最后，无非是两种选择：亏、赢。

延伸一下，其他项目也是如此：

输、赢；正、负；胜、负；涨、跌；加、减；男、女；老、少；黑、白；取、舍。

这些正好对应易经的阴、阳。

计算机的核心是 CPU，但 CPU 的本质不过是一个“加法器”。

在 CPU 里面，其实加、减都是加法，因为减不过是加上一个负数符号。

这个加法器正好对应了易经里面的一生二：一（CPU 加法器）生二（加、减）。

明白了这点再看 Sklearn 的知识图谱，虽然表面看起来还是乱七八糟的，非常烦琐，但仔细梳理后，能够看出其核心其实只有两个字：分类（Type）。

将分类做好，其他匹配、识别都是简单的问题，只是贴上标签和加上备注。

国外也有专家认为，所有的人工智能、机器学习，本质上都是二元一次方程的寻优算法。

笔者在博客中曾经也说过：

> 可以把人工智能、机器学习看成一个巨大的字符串 Find 查找算法，只不过这个算法中的 keyword 关键词，与被查找的字符串的大小非常庞大，趋于无限长度。

众所周知，开源项目 70% 的问题都是卡在系统配置上，特别是对于国外的项目，存在语言障碍，文档不齐。

TopQuant.vip 极宽量化开源社区发布的集成式 Python 开发平台：zwPython，解压缩即用，无须安装；内置人工智能、机器学习所需的各种模块库，包括 NLTK、Sklearn 以及最高大上的 TensorFlow，而且全功能、全免费、全开源，无须花费一分钱，是真正的零起点学 AI。

从某种程度上讲，能够下载 zwPython、解压缩并且运行内置的 hello 程序，相对于传统的人工智能学习周期，大家就已经完成了 70%，并且拥有了专业级别的人工智能开发平台。

很多初学者，面对人工智能、机器学习往往连系统配置都玩不转，最基本的 hello 程序也无法运行。

> 在《zwPython 用户手册》里面，笔者曾经说过，虽然很多人认为 Python 是面向对象的语言，但实践表明，忘记 OOP（面向对象编程）的概念，采用传统的 Basic 语言（面向过程）模式，学习效率可以提高 10 倍以上。
>
> 这一点用过 zwPython 的用户都知道，笔者量化培训班的学员更是深有体会，许多 40 多岁金融一线的从业人员，完全没有编程基础，通常学习两个月，就可以自己修改 zwQuant 量化开源软件中的策略函数。

所以，学习人工智能，笔者的建议就是：从忘却开始，忘却各种乱七八糟的概念。

忘记这些概念之后，就会海阔天空，所谓的人工智能，不过是传统的 Python 函数调用，而且只有寥寥无几的数个函数，在最简单的案例当中，甚至只需要 2~3 个 AI 相关函数。

需要说明的是，对于初学者而言，除了忘却之外，不要左顾右盼，MATLAB、R 语言、Torch、TensorFlow、NLTK……这些都想学，但或许都学不好。

以上这些名词、术语，看起来简单，但其实对于每一个名词术语，若没有一个专业的博士团队，都是无法完成的独立项目。

对于初学者而言，入门阶段就学习一个模块库：Sklearn。

Sklearn 模块库本身就是人工智能、机器学习的行业标准，该有的人工智能、机器学习经典算法它全部都有，其他的模块库无非是在局部进行了某些优化。

至于人工智能的进阶内容，大家不要着急，学会了 Sklearn 的人工智能入门内容后，再看 TensorFlow、Torch 就不会有看天书的感觉了。

5.2　Iris经典爱丽丝

Iris 爱丽丝，是鸢尾属（拉丁学名：Iris L.）的单叶植物，与常见的百合花类似。

Iris 爱丽丝数据集是人工智能、机器学习最经典的数据集，全称是安德森鸢尾花卉数据集，是统计学习的必备数据集。

维基百科有专门的词条：

安德森鸢尾花卉数据集（Anderson’s Iris data set），也称鸢尾花卉数据集（Iris flower data set）或费雪鸢尾花卉数据集（Fisher’s Iris data set），是一类多重变量分析的数据集。它最初是埃德加·安德森从加拿大加斯帕半岛上的鸢尾属花朵中提取的地理变异数据，后由罗纳德·费雪作为判别分析的一个例子，并运用到统计学中。其数据集包含了 50 个样本，都属于鸢尾属下的 3 个亚属，分别是山鸢尾、变色鸢尾和维吉尼亚鸢尾。其 4 个特征被用作样本的定量分析，即花萼和花瓣的长度和宽度。基于这 4 个特征的集合，费雪发展了线性判别分析以确定其属种。

我们的目的就是通过编程，对这 3 个不同种类、150 多组 Iris 爱丽丝植物的数据，采用专业的数据分析手段和人工智能算法，让程序自动判别植物的种类。

从本章开始是人工智能篇，程序编码重新采用 zai 开头，其中 ai 是人工智能的意思。

案例 5-1：Iris爱丽丝

案例 5-1 的文件名是 zai101_iris01.py，核心代码如下：

```
#1
fss='dat/iris.csv'
df=pd.read_csv(fss,index_col=False)
print('\n#1 df')
print(df.tail())
print(df.describe())

#2
d10=df['xname'].value_counts()
print('\n#2 xname')
print(d10)
```

第 1 组代码，运行结果如下：

```
     x1   x2   x3   x4      name
145  6.7  3.0  5.2  2.3  virginica
146  6.3  2.5  5.0  1.9  virginica
147  6.5  3.0  5.2  2.0  virginica
148  6.2  3.4  5.4  2.3  virginica
149  5.9  3.0  5.1  1.8  virginica
             x1          x2          x3          x4
count  150.000000  150.000000  150.000000  150.000000
mean     5.843333    3.054000    3.758667    1.198667
std      0.828066    0.433594    1.764420    0.763161
min      4.300000    2.000000    1.000000    0.100000
25%      5.100000    2.800000    1.600000    0.300000
50%      5.800000    3.000000    4.350000    1.300000
75%      6.400000    3.300000    5.100000    1.800000
max      7.900000    4.400000    6.900000    2.500000
```

第 2 组代码中的函数：

```
d10=df['xname'].value_counts()
```

对植物种类进行了简单的分类统计，共有 3 种，对应的输出信息如下：

```
#2 xname
versicolor    50
setosa       50
virginica     50
Name: xname, dtype: int64
```

由输出信息可以看出，3 种植物名称分别是：山鸢尾（Iris setosa）、变色鸢尾（Iris versicolor）和维吉尼亚鸢尾（Iris virginica）。

看完以上程序，很多人可能觉得过于简单。案例 5-1 作为人工智能的入门程序的确有些过于简单，但很多初学者就卡在这简简单单的入门案例上。

案例 5-1 看起来似乎很简单，但其实做了很多小优化和修改，比如取消了原始的数据列名称，用 x1~x4 代替，与 Top-AI 极宽的 AI 模块兼容，也更加符合人工智能、机器学习的本质。对于机器学习而言，无所谓采用什么名称，在内部都是数组矩阵。

此外，从输出信息来看，统计命令：

```
print(df.describe())
```

在对应的输出信息中没有 xname 的数据，因为 xname 字段是字符串，无法统计分析，需要先对其进行数字化处理，也就是常说的文字信息的矢量化运算。

全程采用 Pandas 学习 Sklearn 人工智能，可以说是笔者的一个微创新，方便初学者把握数据内部的结构与细节。

传统的 Sklearn 人工智能文档，大部分直接采用 Numpy 数组模块，而 Numpy 是为了追求极限性能设计的模块库，很多算法函数非常复杂，不亚于汇编。

从某种程度上讲，绝大部分初学者的人工智能学习之路，在起步阶段就被 Numpy 这个模块库给吓退了。

虽然 Github 项目网站的 Pandas-Sklearn 项目已经启动，但这个项目为了提高效率，采用的是内部耦合，对初学者帮助不大。

本书全部采用现有的 Pandas 命令，从数据源对 Sklearn 进行整合，无须学习额外的语法，更加方便初学者入门。

案例 5-2：爱丽丝进化与文本矢量化

案例 5-2 的文件名是 zai102_iris02.py，将根据 xname 的植物名称，设置一个新的数据字段 xid，来完成这个文本信息的矢量化工作。

案例 5-2 很简单，下面我们分组逐一讲解。

第 1 组代码，读取 Iris 数据文件，并保存到 df 变量：

```
#1
fss='dat/iris.csv'
df=pd.read_csv(fss,index_col=False)
```

第 2 组代码，根据 xname 字段，按 1、2、3 分别设置 xid 字段，完成读取 Iris 爱丽丝数据名称的矢量化操作。xid 数据字段设置为 int 格式，并保存到 iris2.csv 文件中。

```
#2
df.loc[df['xname']=='virginica', 'xid'] = 1
df.loc[df['xname']=='setosa', 'xid'] = 2
df.loc[df['xname']=='versicolor', 'xid'] = 3
df['xid']=df['xid'].astype(int)
df.to_csv('tmp/iris2.csv',index=False)
```

我们已经将 iris2.csv 文件复制到 dat 目录下，dat/iris2.csv，在后面的案例中，大家可以直接使用这个文件作为数据源。

第 3 组代码，输出修改后的 df 数据信息：

```
#3
print('\n3#df')
print(df.tail())
print(df.describe())
```

对应的输出信息是：

```
      x1   x2   x3   x4      xname    xid
145  6.7  3.0  5.2  2.3  virginica    1
146  6.3  2.5  5.0  1.9  virginica    1
147  6.5  3.0  5.2  2.0  virginica    1
148  6.2  3.4  5.4  2.3  virginica    1
149  5.9  3.0  5.1  1.8  virginica    1
               x1         x2         x3         x4        xid
count 150.000000 150.000000 150.000000 150.000000 150.000000
```

```
mean    5.843333   3.054000   3.758667   1.198667    2.000000
std     0.828066   0.433594   1.764420   0.763161    0.819232
min     4.300000   2.000000   1.000000   0.100000    1.000000
25%     5.100000   2.800000   1.600000   0.300000    1.000000
50%     5.800000   3.000000   4.350000   1.300000    2.000000
75%     6.400000   3.300000   5.100000   1.800000    3.000000
max     7.900000   4.400000   6.900000   2.500000    3.000000
```

第 4 组代码，输出 xname 方面的分类统计信息：

```
d10=df['xname'].value_counts()
print('\n3#xname')
print(d10)
```

对应的输出信息是：

```
4#xname
versicolor      50
setosa          50
virginica       50
Name: xname, dtype: int64
```

第 4 组代码，输出 xid 方面的分类统计信息：

```
d10=df['xid'].value_counts()
print('\n4#xid')
print(d10)
```

对应的输出信息是：

```
4#xid
3    50
2    50
1    50
Name: xid, dtype: int64
```

5.3　AI操作流程

本节涉及部分人工智能的理论知识，其中某些内容有些抽象、拗口，如果大家感觉无法细分，属于正常情况，无须纠结，后面的案例会进行具体介绍。

1. 机器学习与测试数据集合

人工智能、机器学习通常都使用两组数据，一组作为训练数据，一组作为测试数据。

在每组数据中都包含一组多维的参数数据集作为特征数据集，以及一组一维的数组作为结果分类数据，从而形成 4 个数据集合。通常这 4 组数据变量的名称是：

- x_train，训练数据，多维参数数据集。
- y_train，训练数据，一维结果数据集。
- x_test，测试数据，多维参数数据集。
- y_test，测试数据，一维结果数据集。

习惯上，train 数据集用于训练，test 数据集用于测试。此外，通过对数据集 y_train 的分析，会生成一个新的预测结果数据集 y_pred，这个数据集也是一维的结果数据集。

通过对结果数据集 y_pred 与实际的测试数据集 y_test 进行对比，就可以检测算法模型的准确度。

2. 机器学习运行流程

通常人工智能、机器学习的算法流程如下：

- 选择模型函数 mx_fun，mx_fun 是我们自定义的机器学习函数接口。
- 把训练用的特征数据集 x_train 和对应的特征（结果）数据集 y_train，输入模型函数 mx_fun。

- 系统内置的机器学习函数，会自动分析特征数据与结果数据之间的关系。这样的一个过程就是机器学习的过程，也是算法建模的过程。
- 通过对训练数据的机器学习和数据分析，系统会生成一个 AI 机器学习模型，我们将其保存到变量 mx。
- 把测试数据 x_test 输入模型变量 mx，mx 会调用内置的分析函数 predict，生成最终的分析结果 y_pred。
- 如果是实盘，输入最新的数据，例如今天的股市数据或正在销售的足彩比赛赔率数据，系统会自动生成相关的预测数据，如：每天或未来几天股市走势数据或比赛输赢结果。

在进行实盘运行之前，我们会对 y_pred 和正确结果数据 y_test 进行对比，以判断模型的准确程度，并通过一些优化措施和结果调整参数进行迭代运算，或者采用其他的模型提高最终结果的准确度。

以下是常用的 AI 运算流程：

选择模型函数 mx_fun → 导入训练数据 → 建立算法模型 MX → 输入测试（实盘）数据 x_test → 调用 predict 分析（预测）函数 → 生成分析（预测）结果 y_pred

3. 经典机器学习算法

目前，人工智能、机器学习虽然光彩夺目，但还处于启蒙阶段，在本书的案例中，会涉及以下经典的机器学习算法。

- 线性回归算法，函数名：LinearRegression。
- 朴素贝叶斯算法，Multinomial Naive Bayes，函数名：Multinomialnb。
- kNN 近邻算法，函数名：KNeighborsClassifier。
- 逻辑回归算法，函数名：LogisticRegression。
- 随机森林算法，Random Forest Classifier，函数名：RandomForestClassifier。
- 决策树算法，Decision Tree，函数名：tree.DecisionTreeClassifier。
- GBDT 迭代决策树算法，Gradient Boosting Decision Tree，又叫 MART

（Multiple Additive Regression Tree），函数名：GradientBoostingClassifier。

- SVM 向量机算法，函数名：SVC。
- SVM-cross 向量机交叉算法，函数名：SVC。

以上算法中的函数名，均为 Sklearn 模块库内置的函数名称，无须使用其他第三方模块库。所以，笔者说 Sklearn 模块库是初学者学习人工智能、机器学习的不二选择。

4. 黑箱大法

以上内容，还只是人工智能、机器学习最简单的部分。大部分初学者，即使克服种种困难，独自完成了人工智能、机器学习开发平台的配置，再面对这些拗口的算法名称，也基本上都会有崩溃的感觉。

其实这种现象很正常，因为这些算法、名称背后都有非常专业的理论和模型，其学术价值和专业难度都不亚于博士学位的研究。

不过，正如笔者前面所说的，初学者学习人工智能、机器学习最好从忘却开始。

同样，面对这些眼花缭乱的专业名称、术语，依然采用忘却的模式，采用黑箱大法，大家无须纠结各种算法背后的理论，只将其看作一个个黑箱函数即可：

```
输入数据 →【黑箱分析】→ 获得结果
```

以上各种算法，我们甚至可以改名为 1 号函数、2 号函数、3 号函数等。

函数调用是 Python 语言的基本功，能够看到这里的读者，相信无论对函数的调用还是编程都已经非常熟悉。

市场经济讲究的是结果导向，对于大部分学习者而言，需要的也只是一个最终结果数据。

如此一来，99%的人工智能理论知识，大家都可以只是简单了解，然后直接调用相关的函数，获取最后的结果即可。

人工智能、机器学习领域的专家学者，也可以采用这种黑箱模式，有了结果数据，再学习理论和算法，也有了具体的数据支持和更多的感性认识，研究过程也会事半功倍。

5.4 数据切割函数

稍后的案例 5-3 会涉及第一个 Sklearn 的专业函数：

```
train_test_split
```

train_test_split 函数只是对数据进行切割，属于数据预处理函数，并非正式的人工智能、机器学习函数。

我们也可以通过其他函数或者自行编写的函数完成类似的功能。在前面的案例中，数据文件的切割也有类似的功能，事实上，足彩版本的足彩数据机器学习建模使用的数据源，正是我们自定义的数据切割函数。

对于普通的小型数据集合而言，Sklearn 内置的 train_test_split 函数使用更加方便。

train_test_split 数据切割函数位于 Sklearn 的 cross_validation 子模块中，功能是从样本中按比例随机选取 train data 和 test data，调用格式一般为：

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.4, random_state=0)
```

其中：

- x 是训练参数的数据集合。
- y 是训练参数 x 对应的结果数据集合。
- test_size 是样本占比，如果是整数，就是样本的数量。
- random_state 是随机数的种子。

案例 5-3：Iris爱丽丝分解

案例 5-3 的文件名是 zai103_iris03.py，具体讲解如何切割相关的数据，下面分组进行介绍。

第 1 组代码，读取 Iris 数据文件，并保存到 df 变量：

```
#1
fss='dat/iris2.csv'
df=pd.read_csv(fss,index_col=False)
```

请注意，这里使用的是我们修改后、增加了 xid 的爱丽丝数据源文件。

第 2 组代码，输出 df 数据信息：

```
#2
print('\n2#df')
print(df.tail())
```

对应的输出信息是：

```
      x1   x2   x3   x4      xname  xid
145  6.7  3.0  5.2  2.3  virginica    1
146  6.3  2.5  5.0  1.9  virginica    1
147  6.5  3.0  5.2  2.0  virginica    1
148  6.2  3.4  5.4  2.3  virginica    1
149  5.9  3.0  5.1  1.8  virginica    1
```

第 3 组代码，根据人工智能、机器学习算法要求，设置总的数据源 x、y：

```
#3
xlst,ysgn=['x1','x2','x3','x4'],'xid'
x,y= df[xlst],df[ysgn]
#
print('\n3# xlst,',xlst)
print('ysgn,',ysgn)
print('x')
print(x.tail())
```

```
    print('y')
    print(y.tail())
```

对应的输出信息是：

```
    3# xlst, ['x1', 'x2', 'x3', 'x4']
    ysgn, xid
    x
          x1   x2   x3   x4
    145  6.7  3.0  5.2  2.3
    146  6.3  2.5  5.0  1.9
    147  6.5  3.0  5.2  2.0
    148  6.2  3.4  5.4  2.3
    149  5.9  3.0  5.1  1.8
    y
    145    1
    146    1
    147    1
    148    1
    149    1
    Name: xid, dtype: int64
```

第 4 组代码，生成 x_train、x_test、y_train、y_test 数据，并输出相关数据的格式信息：

```
    #4
    x_train, x_test, y_train, y_test = train_test_split(x, y,
random_state=1)
    x_test.index.name,y_test.index.name='xid','xid'
    print('\n4# type')
    print('type(x_train),',type(x_train))
    print('type(x_test),',type(x_test))
    print('type(y_train),',type(y_train))
    print('type(y_test),',type(y_test))
```

对应的输出信息是：

```
4# type
type(x_train), <class 'pandas.core.frame.DataFrame'>
type(x_test), <class 'pandas.core.frame.DataFrame'>
type(y_train), <class 'pandas.core.series.Series'>
type(y_test), <class 'pandas.core.series.Series'>
```

第 5 组代码，保存相关的数据：

```
#5
fs0='tmp/iris_'
print('\n5# fs0,',fs0)
x_train.to_csv(fs0+'xtrain.csv',index=False);
x_test.to_csv(fs0+'xtest.csv',index=False)
y_train.to_csv(fs0+'ytrain.csv',index=False,header=True)
y_test.to_csv(fs0+'ytest.csv',index=False,header=True)
```

需要注意的是以下代码：

```
y_test.to_csv(fs0+'ytest.csv',index=False,header=True)
```

其中的 header 参数我们很少使用，这里用于强制保存 xid 字段头信息，不然 y 数据集会缺少字段头 xid，少一行数据，与 x 数据集尺寸不匹配。大家可以自己测试一下，看看 header 值为 False 的结果。

对应的输出信息如下：

```
5# fs0, tmp/iris_
```

需要说明的是，以上数据我们均会复制到 dat 目录中，以用于稍后的案例中。

第 6 组代码，输出 x 数据集：

```
#6
print('\n6# x_train')
print(x_train.tail())
```

```
print('\nx_test')
print(x_test.tail())
```

对应的输出信息如下：

```
6# x_train
      x1   x2   x3   x4
133  6.3  2.8  5.1  1.5
137  6.4  3.1  5.5  1.8
72   6.3  2.5  4.9  1.5
140  6.7  3.1  5.6  2.4
37   4.9  3.1  1.5  0.1

x_test
      x1   x2   x3   x4
xid
128  6.4  2.8  5.6  2.1
114  5.8  2.8  5.1  2.4
48   5.3  3.7  1.5  0.2
53   5.5  2.3  4.0  1.3
28   5.2  3.4  1.4  0.2
```

第 7 组代码，输出 y 数据集：

```
#7
print('\n7# y_train')
print(y_train.tail())
print('\ny_test')
print(y_test.tail())
```

对应的输出信息如下：

```
7# y_train
133    1
137    1
72     3
140    1
```

```
37     2
Name: xid, dtype: int64

y_test
xid
128    1
114    1
48     2
53     3
28     2
Name: xid, dtype: int64
```

案例 5-4：线性回归算法

有了合适的训练数据和测试数据，人工智能真的很简单，只要两三个函数，就可完成相关的人工智能、机器学习编程。

案例 5-4 的文件名为 zai104_iris04.py，是 100%的人工智能、机器学习程序，是人工智能中最常用的线性回归算法，通过对输入数据的学习，自动对测试数据进行分类。

百度百科对应的线性回归词条是：

> 线性回归是利用数理统计中的回归分析，来确定两种或两种以上变量之间相互依赖的定量关系的一种统计分析方法，运用十分广泛。其表达形式为 y = w'x+e，e 为误差，服从均值为 0 的正态分布。

案例 5-4 采用的是线性回归算法，如图 5-2 所示。

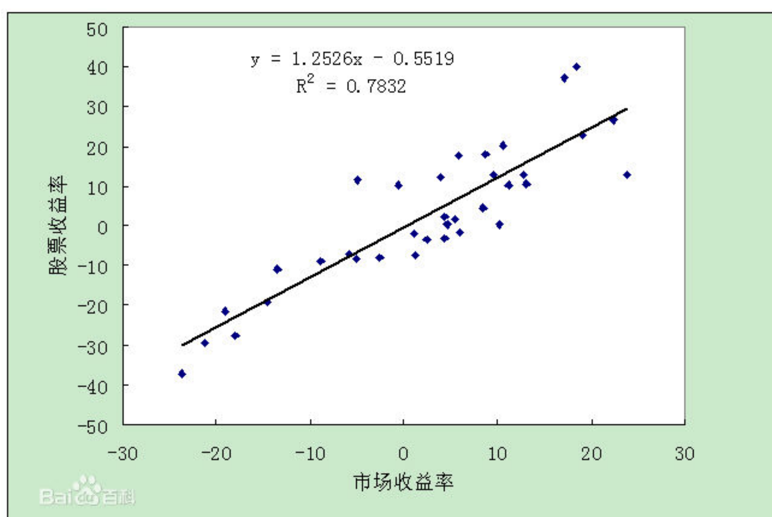


图 5-2　线性回归算法

线性回归算法是最简单、最经典、最古老的人工智能算法，其背后的理论非常复杂，在此，我们采用前面所说的黑箱模式，不予深入讨论，有兴趣的读者请自行参考相关资料。

下面逐一对程序代码进行讲解。

第 1 组代码很简单，读取训练数据，并保存到相关变量：

```
#1
fs0='dat/iris_'
print('\n1# fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
```

第 2 组代码，输出部分训练数据尾部：

```
#2
print('\n2# train')
print(x_train.tail())
print(y_train.tail())
```

对应的输出信息如下：

```
2# train
     x1   x2   x3   x4
107  6.3  2.8  5.1  1.5
108  6.4  3.1  5.5  1.8
109  6.3  2.5  4.9  1.5
110  6.7  3.1  5.6  2.4
111  4.9  3.1  1.5  0.1
    xid
107   1
108   1
109   3
110   1
111   2
```

第 3 组代码，调用机器学习函数，通过对输入数据的分析、学习，建立机器学习模型，并保存到变量 mx：

```
#3
print('\n3# 建模')
mx =zai.mx_line(x_train.values,y_train.values)
```

本案例中第 3 组代码使用的是线性回归算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库中的 LinearRegression 线性回归函数，而是通过 ztop_ai 极宽智能模块库中的 mx_line 函数接口间接进行调用，对应的函数代码是：

```
def mx_line(train_x, train_y):
   mx = LinearRegression()
   mx.fit(train_x, train_y)
   #print('\nlinreg.intercept_')
   #print (mx.intercept_);print (mx.coef_)

   return mx
```

线性回归函数位于 sklearn.linear_model 模块中，函数接口是：

```
    LinearRegression(fit_intercept=True, normalize=False, copy_X=
True, n_jobs=1)
```

mx_line 函数代码很简单，调用基于最小二乘法的 LinearRegression 线性回归函数生成模型变量 mx，运行内置的 fit 命令，分析学习训练数据：train_x（训练数据）、train_y（训练数据对应的答案）。

Sklearn 模块库中的各种机器学习函数，基本上都是采用 fit 命令自动学习、建立模型。

采用 ztop_ai 极宽智能模块库中的 mx_xxx 系列函数，对 Sklearn 模块的各种智能算法函数进行二次封装，其优点是：

- 统一调用接口，规范函数 API 调用模式。
- 无须直接面对底层的机器学习函数，如 LinearRegression，无须了解相关的理论知识即可直接使用。

如果不考虑各种复杂的机器学习函数名称，只需记住 mx_xxx 系列函数，那么建模就基本简化成一个最简单的 fit 内置函数了。

第 4 组代码，读入测试数据并输出相关信息：

```
#4
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
df9=x_test.copy()
print('\n4# x_test')
print(x_test.tail())
```

对应的输出信息是：

```
4# x_test
     x1   x2   x3   x4
33  6.4  2.8  5.6  2.1
34  5.8  2.8  5.1  2.4
35  5.3  3.7  1.5  0.2
```

```
36  5.5  2.3  4.0  1.3
37  5.2  3.4  1.4  0.2
```

第 5 组代码，运行机器学习变量 mx 的内置函数 predict，生成结果数据：

```
#5
print('\n5# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
```

第 6 组代码，读入训练数据的正确答案，并保存到变量 y_test：

```
#6
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
print('\n6# y_test')
print(y_test.tail())
```

对应的输出信息是：

```
6# y_test
    xid
33    1
34    1
35    2
36    3
37    2
```

需要注意的是，本案例为了强调学习，特意把实盘数据放在 predict 函数之后，在生成结果或者预测数据之后再读入，以强调两者之间是完全独立的。

第 7 组代码，整理结果数据变量 df9，并保存到文件中：

```
#7
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n7# df9')
print(df9.tail())
```

对应的输出信息是：

```
7# df9
     x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1  1.551677       1       2
34  5.8  2.8  5.1  2.4  1.209887       1       1
35  5.3  3.7  1.5  0.2  2.093058       2       2
36  5.5  2.3  4.0  1.3  2.317451       3       2
37  5.2  3.4  1.4  0.2  2.300976       2       2
```

大家可以自己打开结果文件，查看相关的结果，其中 y_test 是实盘真实的数据结果，y_pred 是程序生成的结果数据。

第 8 组代码，检验测试结果：

```
#8
dacc=zai.ai_acc_xed(df9,1,False)
print('\n8# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

在本案例中测试结果数据使用的是 ztop_ai 极宽智能模块库中的 zai.ai_acc_xed 函数，这个函数目前有些超前，在后面的章节中再进行详细讲解。

第 8 组代码，对应的输出信息是：

```
8# mx:mx_sum,kok:44.74%
```

由结果数据可以看出，线性回归的准确率有些低，只有 44.74%，不过这个案例是三选一，相比 33%的随机概率还是提高了不少。

至此，一个完整的人工智能、机器学习程序就完成了。虽然有些简陋，但毕竟只是一个开始。众所周知，从 0 到 1 是一个艰难而又漫长的过程，也是一个质变的过程。

“月球的一小步，人类的一大步”。至此，大家迈出了人工智能、机器学习的第一步。

5.5　案例源码

本章包括以下案例程序。

- 案例 5-1：Iris 爱丽丝，文件名是 zai101_iris01.py。
- 案例 5-2：爱丽丝进化与矢量化文本，文件名是 zai102_iris02.py。
- 案例 5-3：Iris 爱丽丝分解，文件名是 zai103_iris03.py。
- 案例 5-4：线性回归算法，文件名是 zai104_iris04.py。

案例 5-1：Iris爱丽丝

文件名是 zai101_iris01.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 Python 程序
@ www.TopQuant.vip      www.ziwang.com

'''

import pandas as pd

#-----------------------

#1
fss='dat/iris.csv'
df=pd.read_csv(fss,index_col=False)
print('\n#1 df')
print(df.tail())
print(df.describe())

#2
```

```
d10=df['xname'].value_counts()
print('\n#2 xname')
print(d10)

#----------------------
print('\nok!')
```

案例 5-2：爱丽丝进化与文本矢量化

文件名是 zai102_iris02.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 Python 程序
@ www.TopQuant.vip     www.ziwang.com

'''

import pandas as pd

#----------------------

#1
fss='dat/iris.csv'
df=pd.read_csv(fss,index_col=False)
#2
df.loc[df['xname']=='virginica', 'xid'] = 1
df.loc[df['xname']=='setosa', 'xid'] = 2
df.loc[df['xname']=='versicolor', 'xid'] = 3
df['xid']=df['xid'].astype(int)
df.to_csv('tmp/iris2.csv',index=False)
```

```
#3
print('\n3#df')
print(df.tail())
print(df.describe())

#4
d10=df['xname'].value_counts()
print('\n4#xname')
print(d10)

#5
d10=df['xid'].value_counts()
print('\n5#xid')
print(d10)



#-----------------------
print('\nok!')
```

案例 5-3：Iris爱丽丝分解

文件名是 zai103_iris03.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 Python 程序
@ www.TopQuant.vip     www.ziwang.com

'''

import pandas as pd

#
```

```
import sklearn
from sklearn.cross_validation import train_test_split
#
#-----------------------

#1
fss='dat/iris2.csv'
df=pd.read_csv(fss,index_col=False)

#2
print('\n2# df')
print(df.tail())



#3
xlst,ysgn=['x1','x2','x3','x4'],'xid'
x,y= df[xlst],df[ysgn]
#
print('\n3# xlst,',xlst)
print('ysgn,',ysgn)
print('x')
print(x.tail())
print('y')
print(y.tail())

#4
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1)
x_test.index.name,y_test.index.name='xid','xid'
print('\n4# type')
print('type(x_train),',type(x_train))
print('type(x_test),',type(x_test))
print('type(y_train),',type(y_train))
print('type(y_test),',type(y_test))

#5
```

```
fs0='tmp/iris_'
print('\n5# fs0,',fs0)
x_train.to_csv(fs0+'xtrain.csv',index=False);
x_test.to_csv(fs0+'xtest.csv',index=False)
y_train.to_csv(fs0+'ytrain.csv',index=False,header=True)
y_test.to_csv(fs0+'ytest.csv',index=False,header=True)

#6
print('\n6# x_train')
print(x_train.tail())
print('\nx_test')
print(x_test.tail())

#7
print('\n7# y_train')
print(y_train.tail())
print('\ny_test')
print(y_test.tail())

#-----------------------
print('\nok!')
```

案例 5-4：线性回归算法

文件名是 zai104_iris04.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统・培训课件-配套教学 Python 程序
@ www.TopQuant.vip      www.ziwang.com

'''

import pandas as pd
```

```
#
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#----------------------

#1
fs0='dat/iris_'
print('\n1# fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);


#2
print('\n2# train')
print(x_train.tail())
print(y_train.tail())


#3
print('\n3# 建模')
```

```
mx =zai.mx_line(x_train.values,y_train.values)

#4
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
df9=x_test.copy()
print('\n4# x_test')
print(x_test.tail())

#5
print('\n5# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred

#6
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
print('\n6# y_test')
print(y_test.tail())


#7
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n7# df9')
print(df9.tail())


#
#8
dacc=zai.ai_acc_xed(df9,1,False)
print('\n8# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#----------------------
print('\nok!')
```

第 6 章 机器学习经典算法案例（上）

前面我们说过，Sklearn 中常用的经典机器学习算法有：线性回归算法、朴素贝叶斯算法、kNN 近邻算法、逻辑回归算法、随机森林算法、决策树算法、GBDT 迭代决策树算法、SVM 向量机算法和 SVM-cross 向量机交叉算法。

在第 5 章我们已经学习过线性回归算法，本章将通过具体的案例，逐一学习相关的人工智能、机器学习经典算法。

需要注意的是，虽然本书和有关文档经常将 Sklearn 中相关的机器学习算法称之为“*xx* 机器学习函数”，但其定义都是 class 类，大家要记住这点。

6.1 线性回归

在 Sklearn 模块库中有多种不同的线性回归函数，都位于 Linear_model 模块中，函数名分别如下。

- Ridge：岭回归算法，如图 6-1 所示。

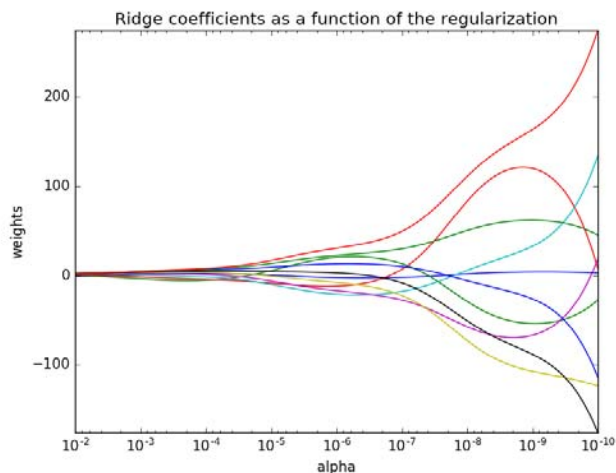


图 6-1　岭回归算法

- LASSO（Least Absolute Shrinkage and Selection Operator）：最小绝对值收缩和选择算法，俗称套索算法，如图 6-2 所示。

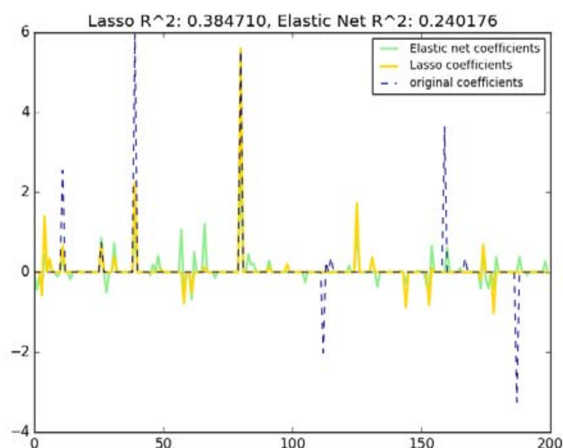


图 6-2　最小绝对值收缩和选择算法

- MultiTaskLasso：多任务 LASSO 回归算法，如图 6-3 所示。
- ElasticNet：弹性网眼算法，如图 6-4 所示。

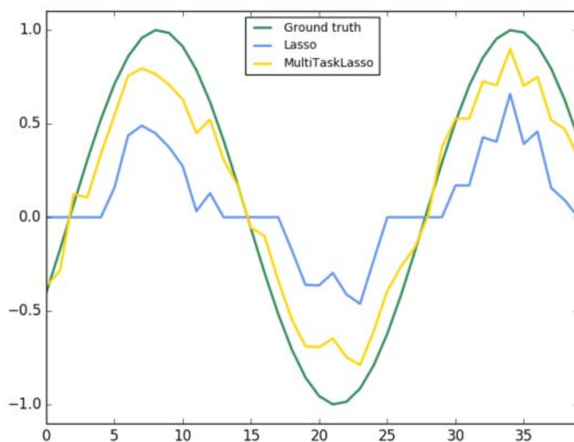


图 6-3　多任务 LASSO 回归算法

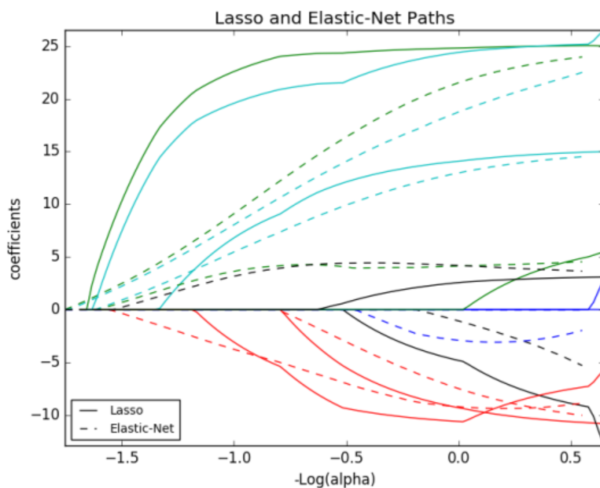


图 6-4　弹性网眼算法

- LassoLars：LARS 套索算法，如图 6-5 所示。
- OrthogonalMatchingPursuit：正交匹配追踪（OMP）算法，如图 6-6 所示。
- BayesianRidge：贝叶斯岭回归算法，如图 6-7 所示。

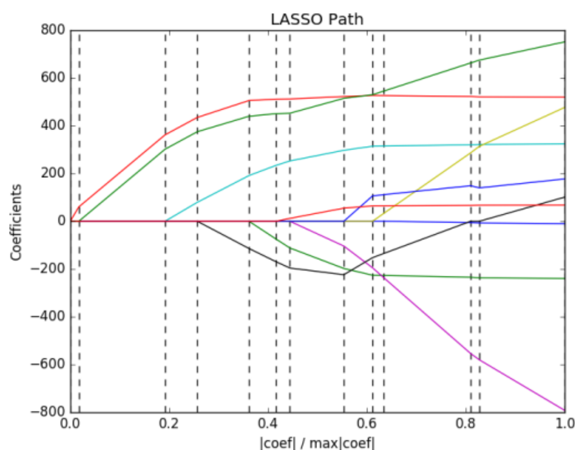


图 6-5　LARS 套索算法

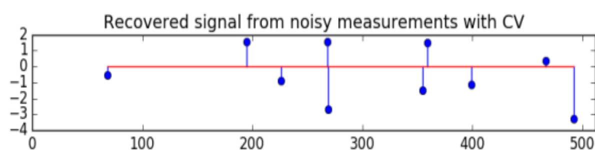


图 6-6　正交匹配追踪（OMP）算法

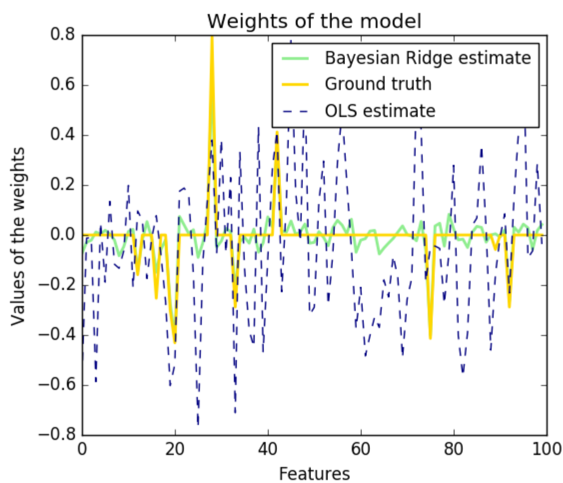


图 6-7　贝叶斯岭回归算法

- ARDRegression：ARD 自相关回归算法，如图 6-8 所示。

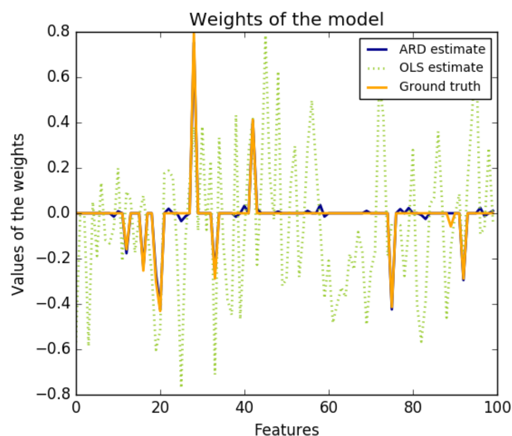


图 6-8　ARD 自相关回归算法

- LogisticRegression：逻辑回归算法，如图 6-9 所示。

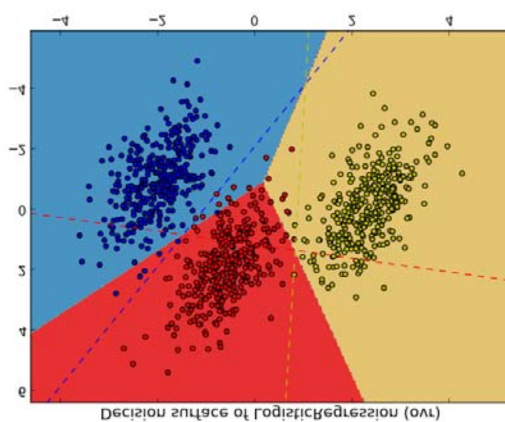


图 6-9　逻辑回归算法

- SGDClassifier：SGD 随机梯度下降算法，如图 6-10 所示。
- MultiTaskElasticNet：多任务弹性网眼算法。
- LARS：最小角回归算法。

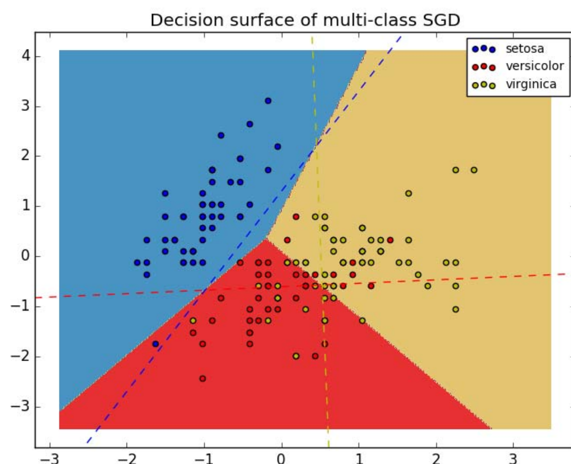


图 6-10　SGD 随机梯度下降算法

- Perceptron：感知器算法，如图 6-11 所示。

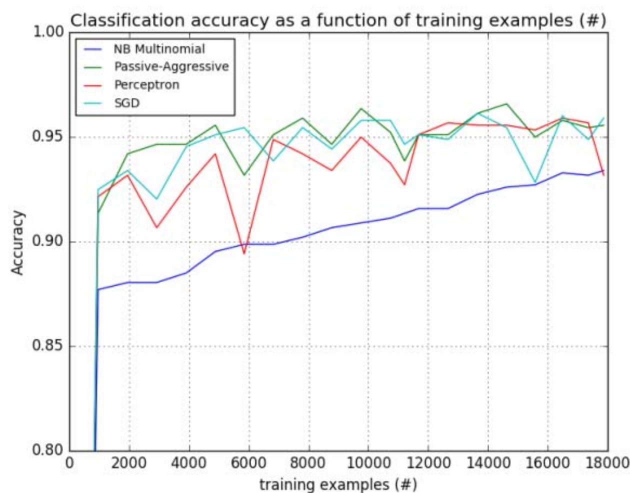


图 6-11　感知器算法

- PassiveAggressiveClassifier：PA 被动感知算法，如图 6-12 所示。
- RANSACRegressor：鲁棒回归算法，如图 6-13 所示。
- HuberRegressor：Huber 回归算法，如图 6-14 所示。

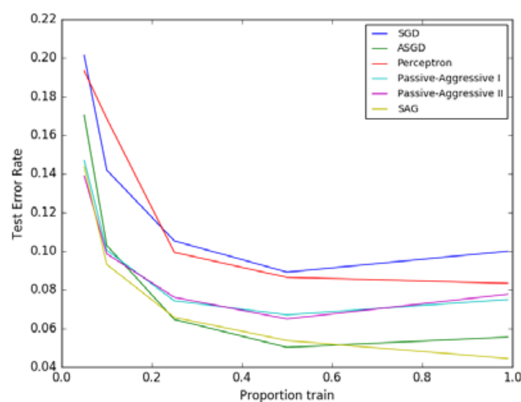


图 6-12　PA 被动感知算法

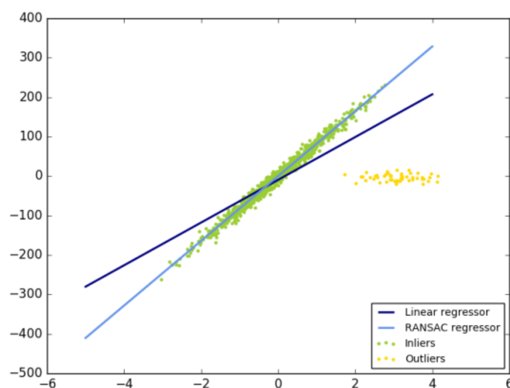


图 6-13　鲁棒回归算法

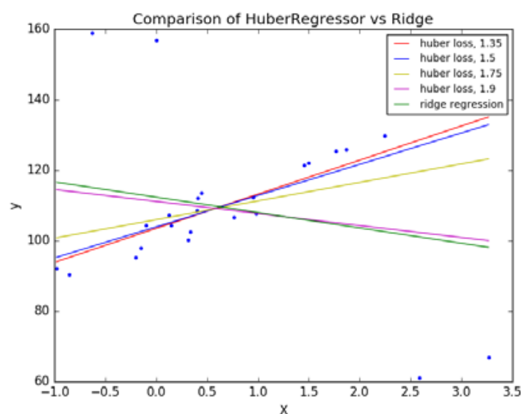


图 6-14　Huber 回归算法

- TheilSenRegressor：Theil-Sen 回归算法，如图 6-15 所示。
- PolynomialFeatures：多项式函数回归算法，如图 6-16 所示。
- LinearRegression：最小二乘法线性回归算法。

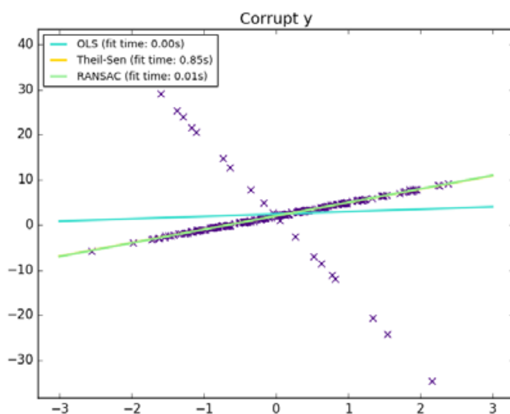


图 6-15　Theil-Sen 回归算法

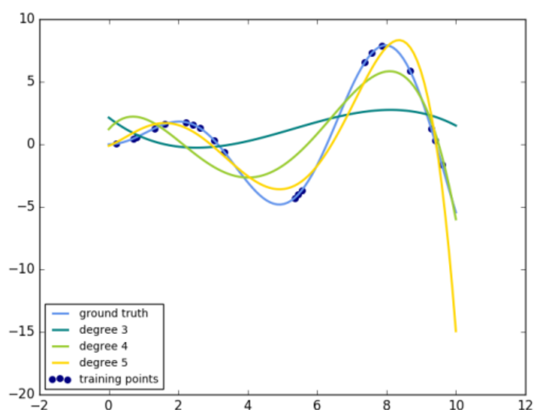


图 6-16　多项式函数回归算法

这些只是 Skearn0.18 版本的 linear_model 模块当中，所包括的机器学习的主要算法函数，关于以上函数需要注意的是：

- LogisticRegression（逻辑回归算法），虽然也属于线性回归算法，但由于用的较多，所以我们把其作为一个独立类别。

- ElasticNet（弹性网眼算法），对弹性神经网络进行分析，是一个改进型的弹性网络算法。
- LASSO 算法与岭回归函数和 LARS 算法很类似。LASSO 算法与岭回归函数都是通过增加惩罚函数来判断、消除特征间的共线性。LASSO 算法与 LARS 算法都可以用作参数选择，得出一个相关系数的稀疏向量。

Sklearn 项目官方网站，有大量的人工智能、机器学习专业文档和源码可以下载，网址是：http://scikit-learn.org。

6.2　逻辑回归算法

逻辑回归算法（LogisticRegression），虽然也属于线性回归算法，但由于使用较多，我们把其作为一个独立类别，如图 6-17 所示。

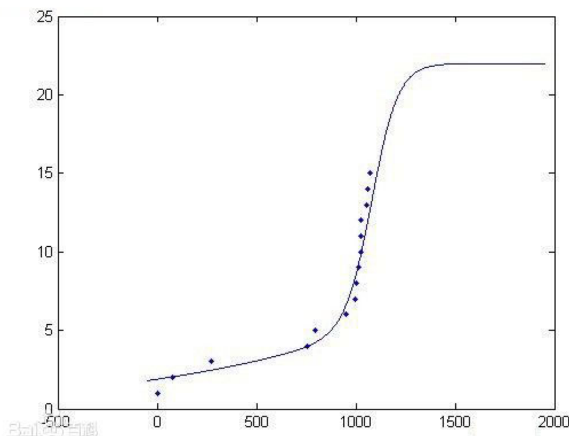


图 6-17　逻辑回归算法

Logistic 回归又称 Logistic 回归分析，是一种广义的线性回归分析模型，常用于数据挖掘、疾病自动诊断、经济预测等领域。例如，探讨引发疾病的危险因素，并根据危险因素预测疾病发生的概率等。

逻辑回归算法函数（LogisticRegression），位于 Sklearn 模块库的 linear_model 线性回归子模块，函数接口是：

```
    LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0,
fit_intercept=True,intercept_scaling=1,class_weight=None,random_s
tate=None, solver='liblinear', max_iter=100, multi_class= 'ovr',
verbose=0, warm_start=False, n_jobs=1)
```

案例 6-1：逻辑回归算法

案例 6-1 文件名是 zai201_mx_log.py，下面将具体介绍逻辑回归算法的应用。

案例 6-1 是基于案例 5-4 线性回归算法，由于案例 5-4 是第一个机器学习案例，为了方便讲解，其中有不少冗余代码，案例 6-1 将其进行了优化，更加接近实盘程序。

下面我们详细讲解案例 6-1 的程序代码。

第 1 组代码，读取训练数据并保存到相关变量，复制 x_test 测试到 df9 结果数据变量中：

```
#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()
```

第 2 组代码，根据逻辑回归算法，建立机器学习模型，并保存到变量 mx：

```
#2
```

```
print('\n2# 建模')
mx =zai.mx_log(x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是逻辑回归算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库当中的 LogisticRegression 逻辑回归建模函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# 逻辑回归算法，函数名，LogisticRegression
def mx_log(train_x, train_y):
    mx = LogisticRegression(penalty='l2')
    mx.fit(train_x, train_y)
    return mx
```

LogisticRegression 回归函数位于 sklearn.linear_model 模块，函数接口在前面已经介绍过了，在此不再赘述。

第 3 组代码，运行机器学习变量 mx 的内置函数 predict，生成结果数据，并保存到结果数据变量 df9：

```
#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)
```

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
     x1   x2   x3   x4  y_predsr  y_test  y_pred
```

```
33  6.4  2.8  5.6  2.1        1      1      1
34  5.8  2.8  5.1  2.4        1      1      1
35  5.3  3.7  1.5  0.2        2      2      2
36  5.5  2.3  4.0  1.3        3      3      3
37  5.2  3.4  1.4  0.2        2      2      2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:84.21%
```

84.21%的结果非常不错了，案例 5-4 线性回归的准确率只有 44.74%：

```
8# mx:mx_sum,kok:44.74%
```

虽然案例 6-1 将程序进行了优化，但还是有不少冗余代码，在实盘操作时只要前面三组代码就完全足够了，而且有些命令还可以进一步精简。

6.3　朴素贝叶斯算法

学过统计学的读者一定都知道贝叶斯定理，这个 250 多年前发明的算法，在信息领域内有着无与伦比的地位。贝叶斯分类是一系列分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。

朴素贝叶斯算法（Naive Bayesian）是其中应用最为广泛的分类算法之一，该算法基于一个简单的假定：给定目标值时，属性之间相互条件独立，如图 6-18 所示。

在 Sklearn 模块库中，涉及贝叶斯原理的算法函数有很多，这里只介绍位于独立子模块 naive_bayes 中的几个相关函数：

- MultinomialNB([alpha, ...])，多项式朴素贝叶斯算法，如图 6-19 所示。

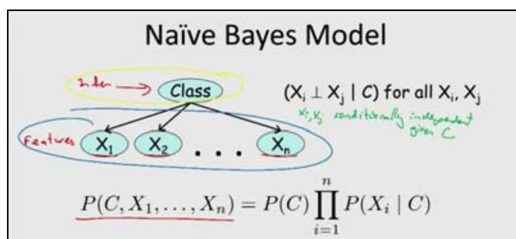


图 6-18　朴素贝叶斯算法



图 6-19　多项式朴素贝叶斯算法

- GaussianNB([priors])，高斯朴素贝叶斯算法，如图 6-20 所示。
- BernoulliNB，伯努利朴素贝叶斯算法。

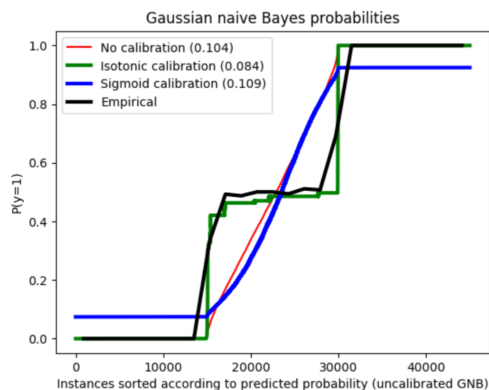


图 6-20　高斯朴素贝叶斯算法

案例 6-2：贝叶斯算法

案例 6-2 文件名是 zai202_mx_nb.py，介绍的是多项式朴素贝叶斯算法，函数名 MultinomialNB 位于 naive_bayes 模块，函数接口是：

```
MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
```

案例 6-2 源码和本章其他小节的源码，基本都与案例 6-1 的源码类似，只是调用的具体的机器学习函数不同，为简化篇幅，本节及后面的章节只介绍程序代码当中不同的部分。

案例 6-2 的核心在于第 2 组代码，建模：

```
#2
print('\n2# 建模')
mx =zai.mx_bayes(x_train.values,y_train.values)
```

本案例中的第 2 组代码，使用的是多项式朴素贝叶斯算法，建立机器学习模型，但是并没有直接调用 Sklearn 模块库当中的 MultinomialNB 建模函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# 多项式朴素贝叶斯算法，Multinomial Naive Bayes，函数名，multinomialnb
def mx_bayes(train_x, train_y):
    mx = MultinomialNB(alpha=0.01)
    mx.fit(train_x, train_y)
    return mx
```

MultinomialNB 回归函数位于 sklearn.naive_bayes 模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
```

```
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
     x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1       1       1
34  5.8  2.8  5.1  2.4         1       1       1
35  5.3  3.7  1.5  0.2         2       2       2
36  5.5  2.3  4.0  1.3         1       3       1
37  5.2  3.4  1.4  0.2         2       2       2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:57.89%
```

57.89%的准确度，结果有些偏低，虽然比线性回归算法的 44.74%的结果好一点，但远远低于逻辑回归算法的 84.21%。

6.4 KNN近邻算法

KNN 近邻算法，又叫作 K 最近邻（KNN，k-NearestNeighbor）分类算法，是数据挖掘分类技术中最简单的方法之一。所谓 K 最近邻就是 k 个最近的邻居的意思，即每个样本都可以用它最接近的 k 个邻居来代表，如图 6-21 所示。

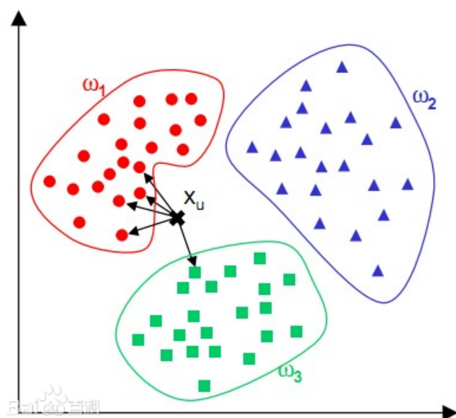


图 6-21　KNN 近邻算法

在 Sklearn 模块库中，KNN 近邻算法相关的算法函数位于 Neighbors 模块库，主要的机器学习算法函数如下。

- KNeighborsClassifier：KNN 近邻算法，如图 6-22 所示。

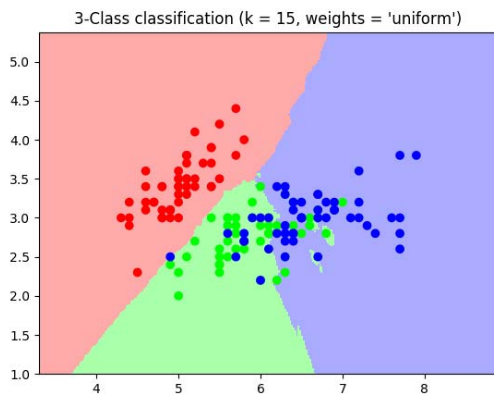


图 6-22　KNN 近邻算法

- NearestNeighbors：最近邻居算法，如图 6-23 所示。
- KNeighborsRegressor：K 近邻回归算法，如图 6-24 所示。
- NearestCentroid：最近质心算法，如图 6-25 所示。

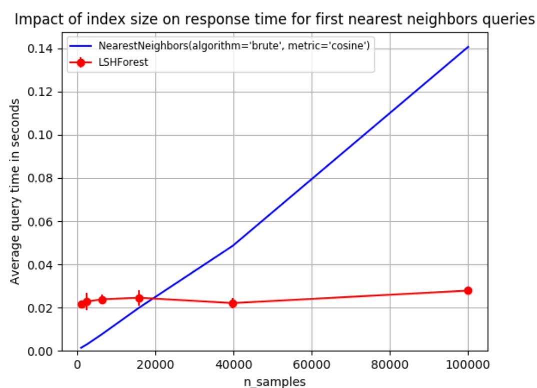


图 6-23　最近邻居算法

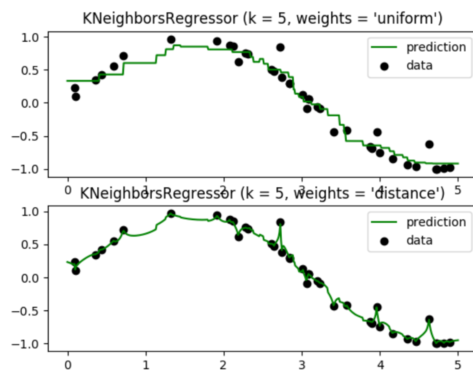


图 6-24　K 近邻回归算法

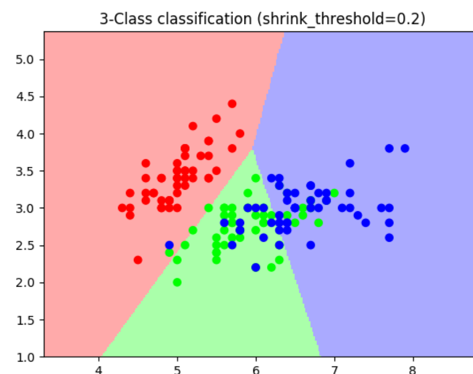


图 6-25　最近质心算法

- LSHForest（Locality Sensitive Hashing forest，LSH）：局部敏感哈希森林算法，是最近邻搜索方法的代替，排序实现二进制搜索和 32 位定长数组和散列，使用 Hash 家族的随机投影方法，近似余弦距离，如图 6-26 所示。

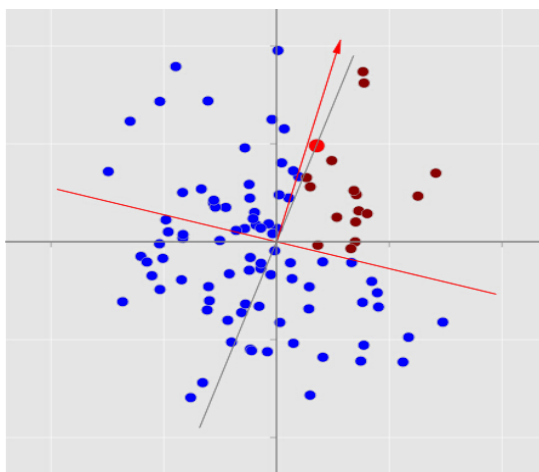


图 6-26　LSH 局部敏感哈希森林算法

案例 6-3：KNN近邻算法

案例 6-3 文件名是 zai203_mx_knn.py，介绍的是 KNN 近邻算法，位于 Neighbors 模块，函数名是 KNeighborsClassifier，函数接口是：

```
    KNeighborsClassifier(n_neighbors=5,weights='uniform',algorith
m='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None,
n_jobs=1, **kwargs)
```

案例 6-3 的核心在于第 2 组代码，建模：

```
#2
print('\n2# 建模')
mx =zai.mx_knn(x_train.values,y_train.values)
```

在案例中，第 2 组代码使用的是 KNN 近邻算法，建立机器学习模型，

但是并没有直接调用 Sklearn 模块库当中的 KNeighborsClassifier 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# KNN 近邻算法，函数名，KNeighborsClassifier
def mx_knn(train_x, train_y):
    mx = KNeighborsClassifier()
    mx.fit(train_x, train_y)
    return mx
```

KNN 近邻算法，位于 sklearn.neighbors 模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
     x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1       1       1
34  5.8  2.8  5.1  2.4         1       1       1
35  5.3  3.7  1.5  0.2         2       2       2
36  5.5  2.3  4.0  1.3         3       3       3
37  5.2  3.4  1.4  0.2         2       2       2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:100.00%
```

这个结果居然是 100%，全垒打，不过大家不要高兴得太早，这个 100%只是基于爱丽丝 Iris 数据集的，数据量有些偏低，不一定有普适性。

不管如何，这个案例也是大家学习人工智能、机器学习算法的第一个 100%准确率的程序，第一个全垒打程序，还是值得庆祝一下的。

6.5　随机森林算法

随机森林算法（Random forest），是指利用多棵树对样本进行训练并预测的一种算法，如图 6-27 所示。随机森林这个术语，是由 1995 年贝尔实验室的 Tin Kam Ho 所提出的随机决策森林（Random Decision Forests）理论发展而来的。后来，Leo Breiman 和 Adele Cutler 推论出了随机森林的算法，并注册了 Random Forests 商标。

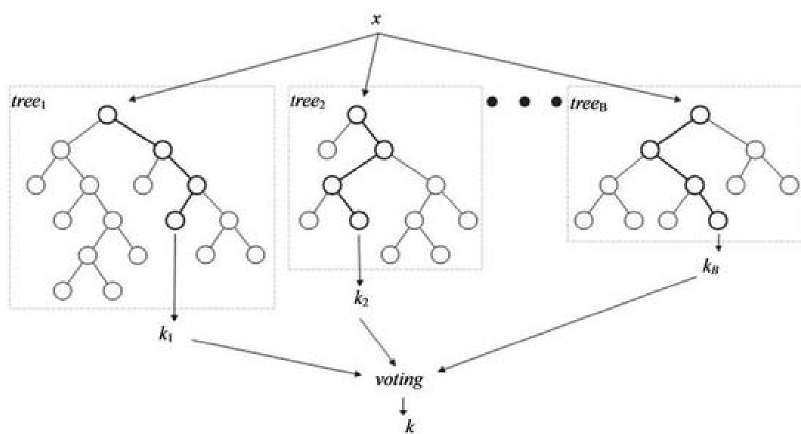


图 6-27　随机森林算法

随机森林算法是一个包含多个决策树的算法，并且其输出的类别是由个别树输出的类别的众数而定，这个算法结合了 Breimans 的“Bootstrap Aggregating”想法和 Tin Kam Ho 的“Random Subspace Method”算法，以建造决策树的集合。

在 Sklearn 模块库中，Random Forest 随机森林算法相关的算法函数，位于集成算法模块 Ensemble 中，其中相关的机器学习算法函数如下。

- RandomForestClassifier：随机森林算法，如图 6-28 所示。

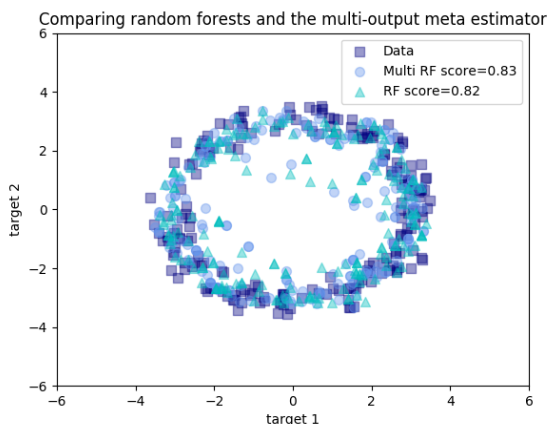


图 6-28　随机森林算法

- BaggingClassifier：Bagging 装袋算法，相当于多个专家投票表决，对于多次测试，每个样本返回的是多次预测结果较多的那个，如图 6-29 所示。

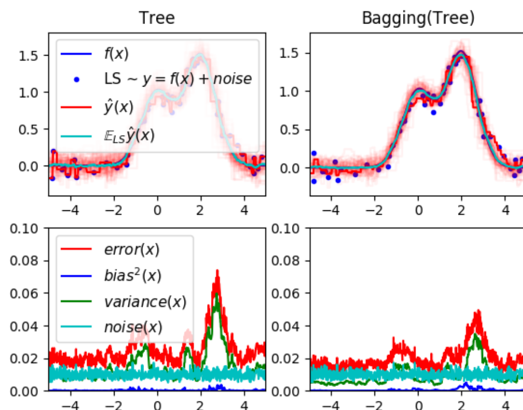


图 6-29　Bagging 装袋算法

- ExtraTreesClassifier：完全随机树算法，如图 6-30 所示。

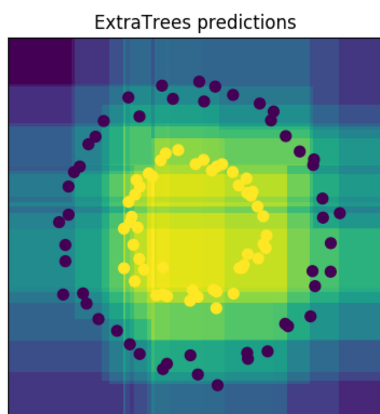


图 6-30　完全随机树算法

- Adaboost：迭代算法，其核心思想是针对同一个训练集，训练不同的分类器（弱分类器），然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器），如图 6-31 所示。

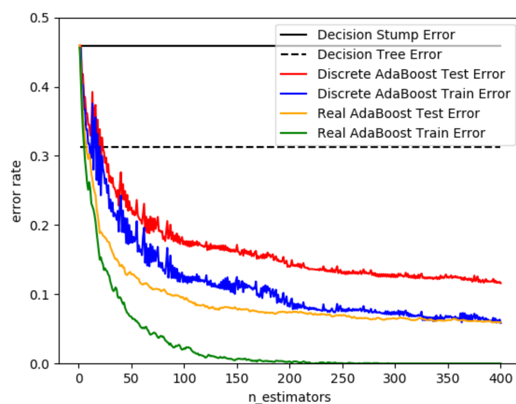


图 6-31　Adaboost 迭代算法

- GradientBoostingClassifier：GBT 梯度 Boosting 树算法，如图 6-32 所示。
- GradientBoostingRegressor：梯度回归算法，如图 6-33 所示。
- VotingClassifier：投票算法，如图 6-34 所示。

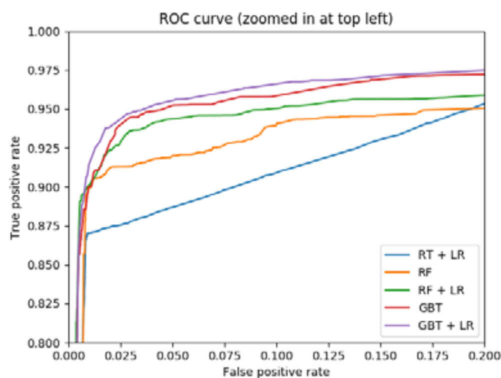


图 6-32　GBT 梯度 Boosting 树算法

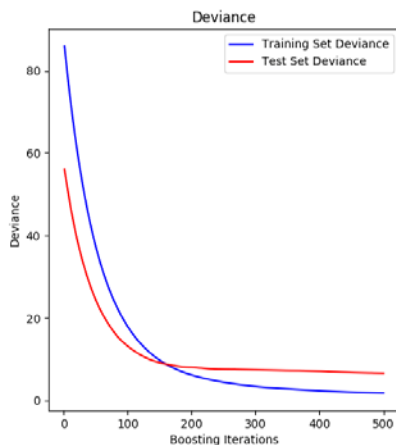


图 6-33　梯度回归算法

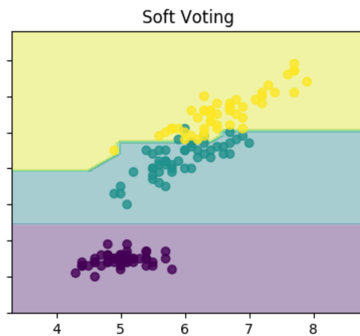


图 6-34　投票算法

案例 6-4：随机森林算法

案例 6-4 文件名是 zai204_mx_rf.py，介绍的是随机森林算法，位于 Ensemble 集成算法模块，函数名是 KNeighborsClassifier，函数接口是：

```
    RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07,bootstrap=True,oob_score=False,n_jobs=1,random_state=None, verbose=0, warm_start=False, class_weight=None)
```

案例 6-4 的核心在于第 2 组代码，建模：

```
    #2
    print('\n2# 建模')
    mx =zai.mx_forest (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是随机森林算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库当中的 RandomForestClassifier 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
    # 随机森林算法, Random Forest Classifier, 函数名,
RandomForestClassifier
    def mx_forest(train_x, train_y):
        mx = RandomForestClassifier(n_estimators=8)
        mx.fit(train_x, train_y)
        return mx
```

随机森林算法位于 sklearn.ensemble 集成算法模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
    #4
    df9.to_csv('tmp/iris_9.csv',index=False)
```

```
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
    x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1        1       1       1
34  5.8  2.8  5.1  2.4        1       1       1
35  5.3  3.7  1.5  0.2        2       2       2
36  5.5  2.3  4.0  1.3        3       3       3
37  5.2  3.4  1.4  0.2        2       2       2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:97.37%
```

结果是 97.37%，这已经非常好了，虽然不是 100%的全垒打，不过作为预测模型，这种精度比 100%给大家的感觉更加心安。

6.6 案例源码

本章包括以下案例程序。

- 案例 6-1：逻辑回归算法，文件名是 zai201_mx_log.py。
- 案例 6-2：朴素贝叶斯算法，文件名是 zai202_mx_nb.py。
- 案例 6-3：KNN 近邻算法，文件名是 zai203_mx_knn.py。
- 案例 6-4：随机森林算法，文件名是 zai204_mx_rf.py。

案例 6-1：逻辑回归算法

案例 6-1：逻辑回归算法，文件名是 zai201_mx_log.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com

'''



import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#-----------------------

#1
```

```
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()

#2
print('\n2# 建模')
mx =zai.mx_log(x_train.values,y_train.values)

#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

案例 6-2：朴素贝叶斯算法

案例 6-2：朴素贝叶斯算法，文件名是 zai202_mx_nb.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com

'''


import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#-----------------------

#1
fs0='dat/iris_'
```

```
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()

#2
print('\n2# 建模')
mx =zai.mx_bayes(x_train.values,y_train.values)

#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#----------------------
print('\nok!')
```

案例 6-3：KNN近邻算法

案例 6-3：KNN 近邻算法，文件名是 zai203_mx_knn.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''



import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta
#
#-----------------------

#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
```

```
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()

#2
print('\n2# 建模')
mx =zai.mx_knn(x_train.values,y_train.values)

#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

案例 6-4：随机森林算法

案例 6-4：随机森林算法，文件名是 zai204_mx_rf.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
```

```
TopQuant-极宽量化系统・培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''

import pandas as pd

import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#-----------------------

#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()
```

```
#2
print('\n2# 建模')
mx =zai.mx_forest(x_train.values,y_train.values)

#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

7

第 7 章 机器学习经典算法案例（下）

7.1 决策树算法

决策树算法最早产生于 20 世纪 60 年代，本质上决策树是通过一系列规则对数据进行分类的过程。

20 世纪 70 年代末，J.Ross Quinlan 提出了 ID3 算法，此算法的目的在于减少树的深度，但是忽略了叶子数目的研究。C4.5 算法在 ID3 算法的基础上进行了改进，在预测变量的缺值处理、剪枝技术、派生规则等方面做了较大改进，既适合于分类问题，又适合于回归问题。

决策树算法是一种逼近离散函数值的方法，也是一种典型的分类方法，其首先对数据进行处理，利用归纳算法生成可读的规则和决策树，然后使用决策对新数据进行分析。决策树算法构造决策树来发现数据中蕴涵的分类规则。如何构造精度高、规模小的决策树是决策树算法的核心内容，如图 7-1 所示。

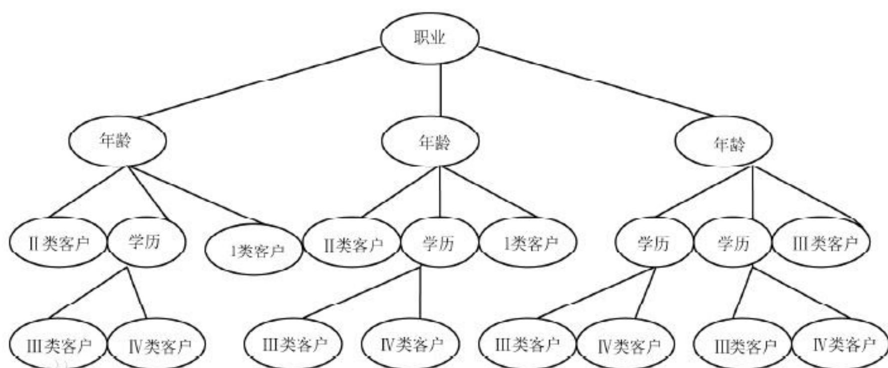


图 7-1　决策树算法

在 Sklearn 模块库中，与 DecisionTreeClassifier 决策树算法相关的算法函数位于 Tree 模块中，其中相关的机器学习算法函数如下。

- DecisionTreeClassifier：决策树算法，如图 7-2 所示。

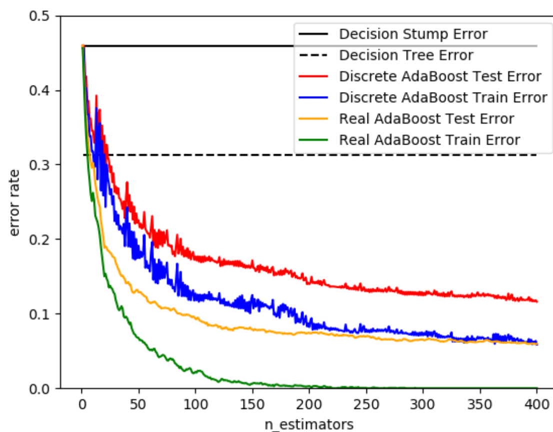


图 7-2　决策树算法

- DecisionTreeRegressor：决策树回归算法，如图 7-3 所示。
- ExtraTreeClassifier：完全随机树算法。
- ExtraTreeRegressor：完全随机树回归算法。
- export_graphviz：辅助函数，输出决策树图形，如图 7-4 所示。

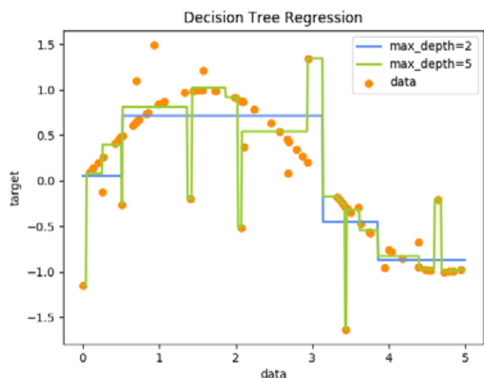


图 7-3　决策树回归算法

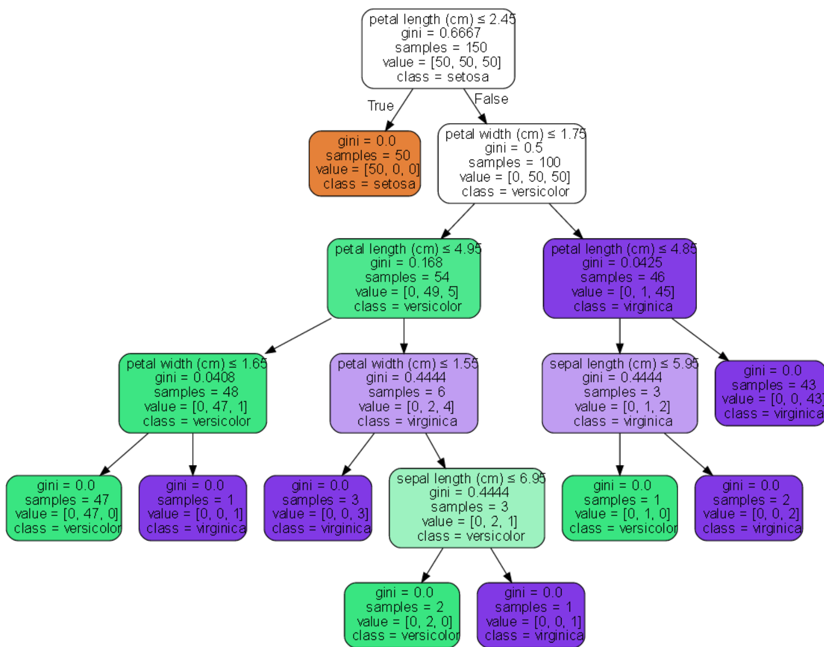


图 7-4　决策树图形

案例 7-1：决策树算法

案例 7-1 文件名是 zai301_mx_dtree.py，介绍的是 Decision Tree 决策树

算法，函数位于 Tree 决策树模块，函数名是 DecisionTreeClassifier，函数接口是：

```
    DecisionTreeClassifier(criterion='gini', splitter='best', max_
depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_
fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_
nodes=None, min_impurity_split=1e-07, class_weight=None, presort=
False)
```

案例 7-1 的核心在于第 2 组代码，建模：

```
#2
print('\n2# 建模')
mx =zai.mx_dtree (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是 Decision Tree 决策树算法建立机器学习模型,但是并没有直接调用 Klearn 模块库当中的 DecisionTreeClassifier 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# 决策树算法，函数名，tree.DecisionTreeClassifier()
def mx_dtree(train_x, train_y):
    mx = tree.DecisionTreeClassifier()
    mx.fit(train_x, train_y)
    return mx
```

Decision Tree 决策树算法位于 sklearn.tree 模块，函数接口在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
     x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1       1       1
34  5.8  2.8  5.1  2.4         1       1       1
35  5.3  3.7  1.5  0.2         2       2       2
36  5.5  2.3  4.0  1.3         3       3       3
37  5.2  3.4  1.4  0.2         2       2       2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:97.37%
```

Decision Tree 决策树算法的测试结果也是 97.37%，这与 KNN 近邻算法的结果一样。

7.2　GBDT迭代决策树算法

目前 GBDT 迭代决策树算法非常火热，因为它近期多次在国际级别的人工智能算法大赛中（如 Kaggle）都获得了胜利。

GBDT 全称为 Gradient Boosting Decision Tree，是一种基于决策树（decision tree）实现的分类回归算法。

GBDT 算法由多棵决策树组成，所有树的结论累加起来做最终结果。它在被提出之初就和 SVM 一起被认为是泛化能力（Generalization）较强的算法。近年更因为被用于搜索排序的机器学习模型而引起大家关注，如图 7-5 所示。

GBDT 是一个应用很广泛的算法，可以用来做分类和回归，在很多的

数据上都有不错的效果。GBDT 这个算法还有一些其他的名字，比如说 MART（Multiple Additive Regression Tree）、GBRT（Gradient Boost Regression Tree）、Tree Net 等。

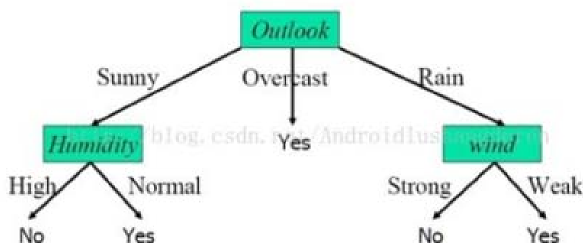


图 7-5　GBDT 算法决策树

案例 7-2：GBDT迭代决策树算法

案例 7-2 文件名是 zai302_mx_gbdt.py，介绍的是 GBDT 迭代决策树算法，函数位于 Ensemble 集成算法模块，函数名是 GradientBoostingClassifier，函数接口是：

```
GradientBoostingClassifier(loss='deviance', learning_rate=0.1,
n_estimators=100, subsample=1.0, criterion='friedman_mse', min_
samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_depth=3, min_impurity_split=1e-07, init=None, random_state=
None, max_features=None, verbose=0, max_leaf_nodes=None, warm_
start=False, presort='auto')
```

案例 7-2 的核心在于第 2 组代码，建模：

```
#2
print('\n2# 建模')
mx =zai.mx_GBDT (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是 GBDT 迭代决策树算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库中的 GradientBoostingClassifier

函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# GBDT 迭代决策树算法, Gradient Boosting Decision Tree,
# 又叫 MART(Multiple Additive Regression Tree)
 def mx_GBDT(train_x, train_y):
    mx = GradientBoostingClassifier(n_estimators=200)
    mx.fit(train_x, train_y)
    return mx
```

GBDT 迭代决策树算法，位于 sklearn.ensemble 集成算法模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
     x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1       1       1
34  5.8  2.8  5.1  2.4         1       1       1
35  5.3  3.7  1.5  0.2         2       2       2
36  5.5  2.3  4.0  1.3         3       3       3
37  5.2  3.4  1.4  0.2         2       2       2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:97.37%
```

无巧不成书，GBDT 迭代决策树算法的测试结果居然也是 97.37%，与 Decision Tree 决策树算法、kNN 近邻算法一样。

7.3 SVM向量机

SVM 全称是 Support Vector Machine，支持向量机，是一种有监督的机器学习算法，通常用来进行模式识别、分类以及回归分析，简称 SVM 向量机，如图 7-6 所示。

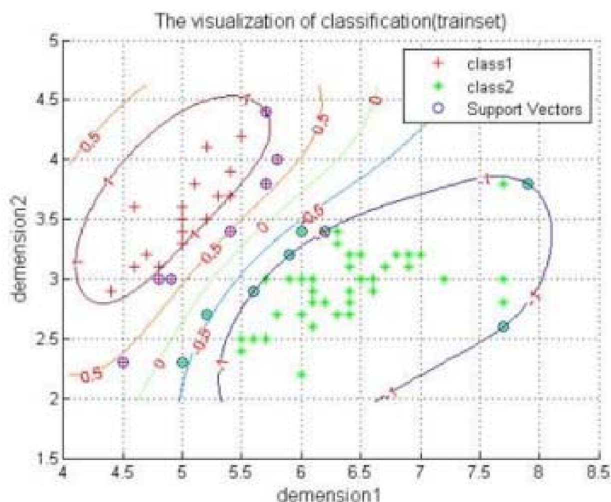


图 7-6 SVM 向量机

在 Sklearn 模块库中，有独立的 SVM 向量机模块 sklearn.svm，其中相关的机器学习算法函数如下。

- SVC：支持向量机算法。
- LinearSVC：线性向量算法。
- NuSVC：Nu 支持向量算法。
- SVR：SVR（TEpsilon）支持向量算法。
- NuSVR：Nu 支持 SVR 向量算法。

- OneClassSVM：一类支持微量机异常检测算法。
- l1_min_c：辅助函数，返回边界参数。

有关函数效果，如图 7-7 所示。

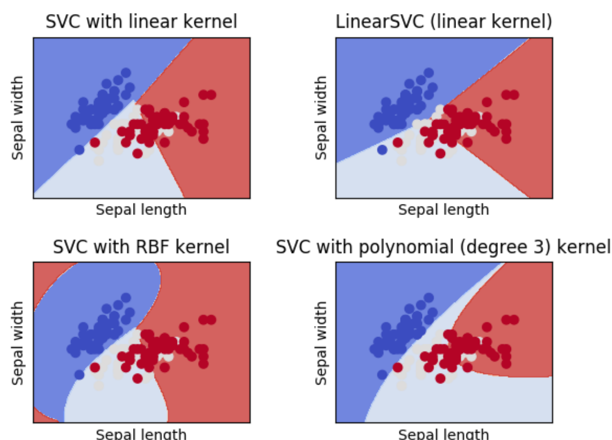


图 7-7　SVM 向量机算法

案例 7-3：SVM向量机算法

案例 7-3 文件名是 zai303_mx_svm.py，介绍的是 SVM 向量机算法，位于 SVM 算法模块，函数名是 KNeighborsClassifier，函数接口是：

```
SVC(C=1.0,kernel='rbf',degree=3,gamma='auto',coef0=0.0,shrinking=True,probability=False,tol=0.001,cache_size=200,class_weight=None,verbose=False,max_iter=-1,decision_function_shape=None,random_state=None)
```

案例 7-3 的核心在于第 2 组代码，建模：

```
#2
print('\n2# 建模')
mx =zai.mx_svm (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是 SVM 向量机算法建立机器学习模型，

但是并没有直接调用 Sklearn 模块库当中的 SVC 向量机函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# SVM 向量机算法，函数名，SVC
def mx_svm(train_x, train_y):
    mx = SVC(kernel='rbf', probability=True)
    mx.fit(train_x, train_y)
    return mx
```

SVM 向量机算法位于 sklearn.svm 模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
     x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1       1       1
34  5.8  2.8  5.1  2.4         1       1       1
35  5.3  3.7  1.5  0.2         2       2       2
36  5.5  2.3  4.0  1.3         3       3       3
37  5.2  3.4  1.4  0.2         2       2       2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:97.37%
```

SVM 向量机算法的结果也是 97.37%。

7.4　SVM-cross向量机交叉算法

SVM-cross 向量机交叉算法，其实也是一种 SVM 向量机算法，函数定义如下：

```
    # SVM- cross 向量机交叉算法，函数名，SVC
    def mx_svm_cross(train_x, train_y):
        mx = SVC(kernel='rbf', probability=True)
        param_grid = {'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000],
'gamma': [0.001, 0.0001]}
        grid_search  =  GridSearchCV(mx,  param_grid,  n_jobs  =  1,
verbose=1)
        grid_search.fit(train_x, train_y)
        best_parameters = grid_search.best_estimator_.get_params()
        #for para, val in best_parameters.items():
        #    print( para, val)
        mx  =  SVC(kernel='rbf',C=best_parameters['C'],gamma=best_
parameters['gamma'], probability=True)
        mx.fit(train_x, train_y)
        return mx
```

SVM- cross 向量机交叉算法函数代码有些复杂，但调用接口都是一样的。

简单来说，SVM- cross 向量机交叉算法，采用了交叉验证（Cross Validation，简称是 CV）模式，在第一次调用 SVC 函数时，根据结果从中选出最好的参数，再一次调用 SVC 函数时，采用类似迭代的方式

交叉验证是用来验证机器学习性能的一种统计分析方法，基本思想是把在某种环境下，将原始数据（Dataset）进行分组，一部分作为训练集（Train Set），另一部分作为验证集（Validation Set），首先用训练集对分类器进行训练，再利用验证集来测试训练得到的模型（Model），以此来作为评价分类器的性能指标。

案例 7-4：SVM-cross向量机交叉算法

案例 7-4 文件名是 zai304_mx_rf.py，介绍的是 SVM-cross 向量机交叉算法。

SVM-cross 向量机交叉算法案例，在某些平台可能会出现运行错误，这个是正常现象，大家知道该函数的调用模式即可。

案例 7-4 的核心在于第 2 组代码，建模：

```
#2
print('\n2# 建模')
mx =zai.mx_svm_cross (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是 SVM-cross 向量机交叉算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库当中的 SVC 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
     x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1       1       1
34  5.8  2.8  5.1  2.4         1       1       1
35  5.3  3.7  1.5  0.2         2       2       2
36  5.5  2.3  4.0  1.3         3       3       3
37  5.2  3.4  1.4  0.2         2       2       2
```

第 5 组代码，检验测试结果：

```
#5
```

```
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:94.74%
```

理论上，SVM-cross 向量机交叉算法的效果应该比 SVM 向量机算法更好，因为毕竟多一道交叉验证迭代运算。

不过案例 7-4 的准确性测试只有 94.74%，低于 SVM 向量机算法的 97.37%，这个也是属于正常。因为有时候会出现过度迭代，这个也从计算流程的角度间接证明了笔者的小数据理论：

> 人工智能、机器学习不是数据量越大越好，也不是迭代计算的次数越多越好。

另外，也因为 Iris 爱丽丝数据集总量太少，才 150 组数据。

7.5　神经网络算法

神经网络算法又被称为神经网络系统（Artificial Neural Networks，简称 ANN），是 20 世纪 40 年代后出现的。它是由众多的神经元可调的连接权值连接而成，具有大规模并行处理、分布式信息存储、良好的自组织自学习能力等特点，如图 7-8 所示。

在神经网络算法中，比较经典的是 BP（Back Propagation）算法，又称为误差反向传播算法，是人工神经网络中的一种监督式的学习算法。BP 神经网络算法在理论上可以逼近任意函数，基本的结构由非线性变化单元组成，具有很强的非线性映射能力。而且网络的中间层数、各层的处理单元数及网络的学习系数等参数可以根据具体情况设定，灵活性很大，在优化、信号处理与模式识别、智能控制、故障诊断等许多领域都有着广泛的应用前景。

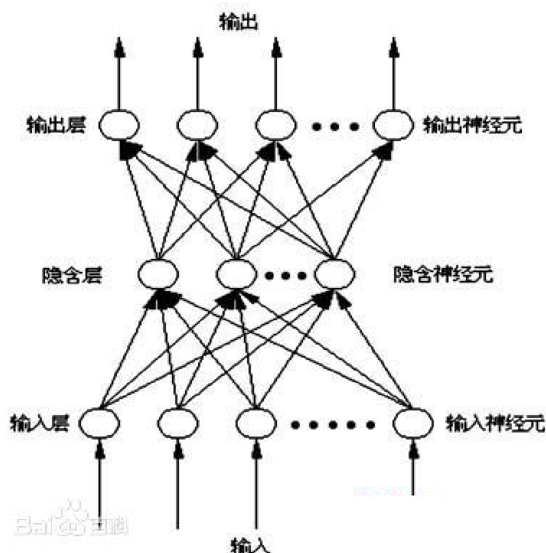


图 7-8　神经网络算法

1. 经典神经网络算法

常用的神经网络算法如下。

- BP 神经网络算法：又称为误差反向传播算法，是人工神经网络中的一种监督式的学习算法。
- RBF（径向基）神经网络算法：20 世纪 80 年代末，由 J.Moody 和 C.Darken 提出，是基于径向基函数（RBF-Radial Basis Function）的神经网络算法，RBF 网络是一种局部逼近网络，能够以任意精度逼近任意连续函数，特别适合于解决分类问题。
- 感知器神经网络算法：是一个具有单层计算神经元的神经网络，网络的传递函数是线性阈值单元。
- 线性神经网络算法：是一种比较简单的神经网络算法，由一个或者多个线性神经元构成。采用线性函数作为传递函数，所以可以输出任意值。
- 自组织神经网络算法：芬兰学者 Kohonen 根据生物神经细胞中存在的自学习特征，提出了自组织特征映射神经网络模型。他认为一个

神经网络在接受外界输入模式时，会自适应地对输入信号的特征进行学习，进而自组织成不同的区域，并且在各个区域对输入模式具有不同的响应特征。包括自组织竞争网络算法、自组织特征映射网络算法、学习向量化网络算法等。

- 反馈神经网络算法：该理论认为，信息在前向传递的同时还要进行反向传递，这种信息的反馈可以发生在不同网络层的神经元之间，也可以只局限于某一层神经元上。反馈网络的典型代表是 Elman 网络算法和 Hopfield 网络算法。

目前最热门的是 Deep learning 深度学习算法，AlphaGo、TensorFlow 主打的都是深度学习算法。

以下是百度百科关于深度学习的词条：

深度学习的概念源于人工神经网络的研究，包含多隐层的多层感知器就是一种深度学习结构。深度学习通过组合低层特征形成更加抽象的高层，表示属性类别或特征，以发现数据的分布式特征表示。

深度学习的概念由 Hinton 等人于 2006 年提出。基于深度置信网络（DBN）提出非监督贪心逐层训练算法，为解决深层结构相关的优化难题带来希望，随后提出多层自动编码器深层结构。此外 Lecun 等人提出的卷积神经网络是第一个真正的多层结构学习算法，它利用空间的相对关系减少参数数目，以提高训练性能。

深度学习是机器学习研究中的一个新领域，其动机在于建立模拟人脑进行分析学习的神经网络，模仿人脑的机制来解释数据，例如图像、声音和文本。

同机器学习方法一样，深度机器学习方法也有监督学习与无监督学习之分，不同的学习框架下建立的学习模型也有很大不同，例如，卷积神经网络（Convolutional neural networks，简称 CNNs）就是一种深度的监督学习下的机器学习模型，而深度置信网（Deep Belief Nets，简称 DBNs）就是一种无监督学习下的机器学习模型。

2. Sklearn神经网络算法

在 Sklearn0.17 版本以前，其对于神经网络算法的支持很弱，通常都是使用其兼容模块库 SKNN 代替，或者使用其他的 Python 模块库，比如 Theano、Pybrain、FFNN、Pylearn2 等。

不过随着神经网络算法在人工智能、机器学习领域的异军突起，特别是 AlphaGo、TensorFlow、Torch 主打的都是深度学习算法 Deep learning，所以 Sklearn 模块库自 0.18 版本开始，也强化了神经网络的功能，目前 sklearn.neural_network 神经网络模块提供了以下 3 种算法函数。

- BernoulliRBM：伯努利受限玻尔兹曼机神经网络算法，简称 RBM 算法。
- MLPClassifier：多层感知器神经网络算法，简称 MLP 算法。
- MLPRegressor：多层感知器神经网络回归算法，简称 MLP 回归算法。

3. 人工智能学习路线图

很多初学者面对各种人工智能算法眼花缭乱，无所适从。

随着 Sklearn 模块库的完善，特别是神经网络算法模块的加入，目前学习人工智能、机器学习相对简单很多，初学者应直接从 Sklearn 开始学习，无须再借助其他模块库进行配套学习。

熟练掌握了 Sklearn 以后，就可以直接学习 TensorFlow 深度学习的知识内容。

以下是百度百科对于 TensorFlow 的介绍：

> TensorFlow是谷歌基于DistBelief进行研发的第二代人工智能学习系统，其命名来源于本身的运行原理。
>
> Tensor（张量）意味着 *N* 维数组，Flow（流）意味着基于数据流图的计算，TensorFlow 为张量从流图的一端流动到另一端的计算过程。TensorFlow 是将复杂的数据结构传输至人工智能神经网进行分析和处理过程的系统。

TensorFlow 表达了高层次的机器学习计算，大幅简化了第一代系统，并且具备更好的灵活性和可延展性。TensorFlow 一大亮点是支持异构设备分布式计算，能够在各个平台上自动运行模型，从手机、单个 CPU / GPU 到成百上千 GPU 卡组成的分布式系统。TensorFlow 支持 CNN、RNN 和 LSTM 算法，这都是目前在 Image、Speech 和 NLP 最流行的深度神经网络模型。

案例 7-5：MLP神经网络算法

MLP（Multilayer Perceptron）神经网络算法，又称多层感知器神经网络算法，对应的函数名是 MLPClassifier。

多层感知器也叫人工神经网络（Artificial Neural Network，ANN），除了输入、输出层，中间可以有多个隐层，最简单的 MLP 只包含一个隐层，即三层的结构，如图 7-9 所示。

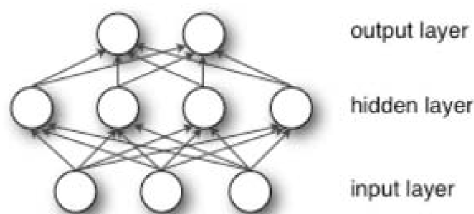


图 7-9　MLP 多层感知器神经网络算法

由图 7-9 可以看出，多层感知器层与各层之间是全连接的（全连接的意思就是：上一层的任何一个神经元与下一层的所有神经元都有连接）。多层感知器最底层是输入层，中间是隐藏层，最上面是输出层。

案例 7-5 文件名是 zai305_mx_MLP.py，介绍的是 MLP 神经网络算法，函数位于 neural_network 神经网络模块，函数名是 MLPClassifier，函数接口是：

```
    MLPClassifier(hidden_layer_sizes=(100,),activation='relu',sol
ver='adam',alpha=0.0001,batch_size='auto',learning_rate='constant
',learning_rate_init=0.001,power_t=0.5, max_iter=200, shuffle=True,
random_state=None,tol=0.0001,verbose=False,warm_start=False,momen
tum=0.9,nesterovs_momentum=True,early_stopping=False,validation_f
raction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```

案例 7-5 的核心在于第 2 组代码，建模：

```
#2
print('\n2# 建模')
mx =zai.mx_MLP (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是 MLP 神经网络算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库当中的 MLPClassifier 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# MLP 神经网络算法
def mx_MLP(train_x, train_y):
    #mx  =  MLPClassifier(solver='lbfgs',  alpha=1e-5,hidden_
layer_sizes=(5, 2), random_state=1)
    mx = MLPClassifier()
    mx.fit(train_x, train_y)
    return mx
```

MLP 神经网络算法位于 neural_network 神经网络模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

有趣的是，MLP 神经网络算法每次运行的结果都有所不同，大家可以多运行几次试一下，对应的输出信息是：

```
4# df9
    x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1       1       1
34  5.8  2.8  5.1  2.4         1       1       1
35  5.3  3.7  1.5  0.2         2       2       2
36  5.5  2.3  4.0  1.3         3       3       3
37  5.2  3.4  1.4  0.2         2       2       2
```

再一次运行的结果是

```
4# df9
    x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1       1       1
34  5.8  2.8  5.1  2.4         1       1       1
35  5.3  3.7  1.5  0.2         2       2       2
36  5.5  2.3  4.0  1.3         3       3       3
37  5.2  3.4  1.4  0.2         2       2       2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

因为 MLP 神经网络算法每次运行的结果不一定相同，所以准确度测试信息是：

```
5# mx:mx_sum,kok:89.47%
```

还有：

```
5# mx:mx_sum,kok:94.74%
```

这种动态的结果，可能是 MLP 神经网络算法内部采用了不同的随机种子，以及计算流程的不同，从而造成最终的结果有误差。

另外，案例 7-5 的核心程序，第 2 组建模代码：

```
#2
```

```
print('\n2# 建模')
mx =zai.mx_MLP (x_train.values,y_train.values)
```

可以发现，尽管神经网络算法与其他的机器学习算法相比，无论是算法理论，还是具体实现都完全不同，但由于我们采用的都是 ztop_ai 极宽智能模块库的统一接口，使用完全一样，所以没有任何额外的学习成本。

案例 7-6：MLP_reg神经网络回归算法

案例 7-6 文件名是 zai306_mx_MLP_reg.py，介绍的是 MLP_reg 神经网络回归算法（又称多层感知器神经网络回归算法），函数位于 neural_network 神经网络模块，函数名是 MLPRegressor，函数接口是：

```
MLPRegressor(hidden_layer_sizes=(100,  ),  activation='relu', solver='adam',  alpha=0.0001,  batch_size='auto',  learning_rate='constant',  learning_rate_init=0.001,  power_t=0.5,  max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```

案例 7-6 的核心在于第 2 组代码，建模：

```
#2
print('\n2# 建模')
mx =zai.mx_MLP_reg(x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是 MLP_reg 神经网络回归算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库当中的 MLPRegressor 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# MLP 神经网络回归算法
def mx_MLP_reg(train_x, train_y):
    #mx  =  MLPClassifier(solver='lbfgs',  alpha=1e-5,hidden_
```

```
layer_sizes=(5, 2), random_state=1)
       mx = MLPRegressor()
       mx.fit(train_x, train_y)
       return mx
```

MLP_reg 神经网络回归算法位于 neural_network 神经网络模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

有趣的是，MLPreg 神经网络回归算法每次运行的结果也都有所不同，大家可以多运行几次试一下，对应的输出信息是：

```
4# df9
    x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1  2.470013       1       2
34  5.8  2.8  5.1  2.4  2.393832       1       2
35  5.3  3.7  1.5  0.2  1.601259       2       2
36  5.5  2.3  4.0  1.3  1.969691       3       2
37  5.2  3.4  1.4  0.2  1.572605       2       2
```

再一次运行的结果是：

```
4# df9
    x1   x2   x3   x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1  2.307966       1       2
34  5.8  2.8  5.1  2.4  2.349912       1       2
35  5.3  3.7  1.5  0.2  1.898731       2       2
36  5.5  2.3  4.0  1.3  1.846610       3       2
37  5.2  3.4  1.4  0.2  1.809671       2       2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

因为 MLPreg 神经网络回归算法，每次运行的结果不一定相同，所以准确度测试信息是：

```
5# mx:mx_sum,kok:34.21%
```

还有：

```
5# mx:mx_sum,kok:28.95%
```

这种动态的结果，也可能是神经网络回归算法内部采用了不同的随机种子，从而造成最终的结果有误差。

28.95%~34.21%的准确度的确有点低，不过这正好是人工三选一的随机概率，这其实也是一个很有趣的课题。

虽然这个算法的准确度较低，但如果这个准确度，真是神经网络算法造成的，那么在人性化模拟方面，也是非常成功的。

7.6 案例源码

本章包括以下案例程序。

- 案例 7-1：决策树算法，文件名是 zai301_mx_dtree.py。
- 案例 7-2：GBDT 迭代决策树算法，文件名是 zai302_mx_gbdt.py。
- 案例 7-3：SVM 向量机算法，文件名是 zai303_mx_svm.py。
- 案例 7-4：SVM- cross 向量机交叉算法，文件名是 zai304_mx_svmcr.py。
- 案例 7-5：MLP 神经网络算法，文件名是 zai305_mx_mlp.py。
- 案例 7-6：MLP_reg 神经网络回归算法，文件名是 zai306_mx_mlp_re.py。

案例 7-1：决策树算法

案例 7-1：决策树算法，文件名是 zai301_mx_dtree.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''


import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#-----------------------

#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
```

```
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()

#2
print('\n2# 建模')
mx =zai.mx_dtree(x_train.values,y_train.values)

#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

案例 7-2：GBDT迭代决策树算法

案例 7-2：GBDT 迭代决策树算法，文件名是 zai302_mx_gbdt.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com

'''


import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#-----------------------

#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
```

```
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()

#2
print('\n2# 建模')
mx =zai.mx_GBDT(x_train.values,y_train.values)

#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

案例 7-3：SVM向量机算法

案例 7-3：SVM 向量机算法，文件名是 zai303_mx_svm.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
```

```
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com

'''


import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#-----------------------

#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()
```

```
#2
print('\n2# 建模')
mx =zai.mx_svm(x_train.values,y_train.values)

#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

案例 7-4：SVM- cross向量机交叉算法

案例 7-4：SVM- cross 向量机交叉算法，文件名是 zai304_mx_svmcr.py，源码如下，SVM- cross 向量机交叉算法案例，在某些平台可能会出现运行错误，这个是正常现象，大家知道该函数的调用模式即可。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
```

```
@ www.TopQuant.vip     www.ziwang.com

'''

import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#-----------------------

#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()

#2
```

```
print('\n2# 建模')
mx =zai.mx_svm_cross(x_train.values,y_train.values)

#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

案例 7-5：MLP神经网络算法

案例 7-5：MLP 神经网络算法，文件名是 zai305_mx_mlp.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com

```

```
'''
import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#----------------------

#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()

#2
print('\n2# 建模')
mx =zai.mx_MLP(x_train.values,y_train.values)
```

```
#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

案例 7-6：MLP_reg神经网络回归算法

案例 7-6：MLP_reg 神经网络回归算法，文件名是 zai306_mx_mlp_re.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''

import pandas as pd
```

```
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import ztop_ai as zai
import zpd_talib as zta

#
#----------------------

#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
df9=x_test.copy()

#2
print('\n2# 建模')
mx =zai.mx_MLP_reg(x_train.values,y_train.values)

#3
print('\n3# 预测')
```

```
y_pred = mx.predict(x_test.values)
df9['y_predsr']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_predsr']).astype(int)

#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())

#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

#-----------------------
print('\nok!')
```

第 8 章 机器学习组合算法

前面几章，我们已经学习了多种常用的机器学习算法，本章将介绍机器学习的组合算法，就是集成多种不同的机器学习算法，从而提高预测结果的准确度。

8.1 CCPP数据集

在介绍机器学习组合算法以前，我们介绍一个新的经典数据集 CCPP 数据集。CCPP 是英文 Combined Cycle Power Plant 的缩写，是联合循环电厂的意思。

CCPP 数据集源自企业的真实数据，是美国联合循环电厂 2006—2011 年，超过 6 年的满负荷工作记录，共包含 9568 条数据，是以小时为单位的检测数据，每条数据均包括以下字段。

- AT：温度。
- V：压力。
- AP：湿度
- RH：压强。

- PE：电力输出。

我们采用人工智能、机器算法的目的，就是根据前面四组字段：AT 温度、V 压力、AP 湿度、RH 压强，来建立机器学习模型，预测 PE 电力输出的数值。

CCPP 数据集也是美国加州大学欧文分校（University of California, Irvine，简称 UC Irvine 或 UCI）机器学习公开课的课件数据。

CCPP 数据集原始文档是 Excel 格式，我们将其改为 CSV 格式，以便于 Pandas 处理，文件名是：dat/ccpp.csv。

在 Sklearn 模块内部已经内置了大量的数据集，包括前面介绍的 Iris 爱丽丝经典数据集。我们之所以使用外部数据集，而不直接调用 Sklearn 模块的内部数据集，是因为该模块问世较早，当时还没有 Pandas 数据分析模块，所以数据集使用的是 Numpy 格式，使用很不方便。

此外，使用 CCPP 数据集取代 Iris 爱丽丝数据集还有以下因素：

- Iris 爱丽丝数据集规模太小，只有 150 条数据，在性能测试方面无法体现各种机器学习算法的差异。
- CCPP 数据集有近万条数据，规模适中，既便于学习，也可以反映出各种机器学习算法的性能差距，如准确度、运行时间等。
- Iris 爱丽丝数据集预测的结果是分类数据，就是三个种类，CCPP 数据集的预测结果 PE 需要输出具体的数值，这是机器学习预测结果的两种模式，两个数据集刚好可以互补。

案例 8-1：CCPP数据集

案例 8-1 文件名是：zai401_ccpp01.py，介绍 CCPP 数据集的基本情况。

案例 8-1 很简单，主要代码如下：

```
fss='dat/ccpp.csv'
```

```
df=pd.read_csv(fss,index_col=False)
print('\n#1 df')
print(df.tail())
print(df.describe())
```

运行结果如下：

```
#1 df
        AT     V      AP     RH     PE
9563  15.12  48.92  1011.80  72.93  462.59
9564  33.41  77.95  1010.30  59.72  432.90
9565  15.99  43.34  1014.20  78.66  465.96
9566  17.65  59.87  1018.58  94.65  450.93
9567  23.68  51.30  1011.86  71.24  451.67
               AT            V           AP           RH           PE
count 9568.000000 9568.000000 9568.000000 9568.000000 9568.000000
mean  19.651231   54.305804   1013.259078 73.308978   454.365009
std   7.452473    12.707893   5.938784    14.600269   17.066995
min   1.810000    25.360000   992.890000  25.560000   420.260000
25%   13.510000   41.740000   1009.100000 63.327500   439.750000
50%   20.345000   52.080000   1012.940000 74.975000   451.550000
75%   25.720000   66.540000   1017.260000 84.830000   468.430000
max   37.110000   81.560000   1033.300000 100.160000  495.760000
```

由输出结果可以看出：

- CCPP 数据集共有 9568 条记录。
- AT 温度的平均数值是 19.65，最小值是 1.81，最大值是 37.11，中位数是 20.34，方差值是 7.45。
- V 压力的平均数值是 54.3，最小值是 25.36，最大值是 81.56，中位数是 52.08，方差值是 12.7。
- AP 湿度的平均数值是 1013.25，最小值是 992.89，最大值是 1033.3，中位数是 1012.94，方差值是 5.94。

- RH 压强的平均数值是 73.3，最小值是 25.56，最大值是 100.16，中位数是 74.98，方差值是 14.6。
- 最终结果数据，PE 电力输出的平均数值是 454.37，最小值是 420.25，最大值是 495.76，中位数是 451.55，方差值是 17.07。

案例 8-2：CCPP数据切割

因为 Skearn 的数据切割函数每次切割的结果都可能不同，所以为了保持学习体验一致，我们采用统一的训练数据集和测试数据集。

案例 8-2 文件名是 zai402_ccpp02.py，介绍 CCPP 数据集的切割算法，以及切割后的各个数据文件的基本情况。

案例 8-2 的程序很简单，下面我们分组进行介绍。

第 1 组、第 2 组代码，读取 CCPP 数据文件保存到 df 变量，并输出数据尾部信息：

```
#1
fss='dat/ccpp.csv'
df=pd.read_csv(fss,index_col=False)

#2
print('\n2# df')
print(df.tail())
```

对应的输出信息是：

```
2# df
          AT      V       AP     RH      PE
9563  15.12  48.92  1011.80  72.93  462.59
9564  33.41  77.95  1010.30  59.72  432.90
9565  15.99  43.34  1014.20  78.66  465.96
9566  17.65  59.87  1018.58  94.65  450.93
9567  23.68  51.30  1011.86  71.24  451.67
```

第 2 组代码是程序的核心所在，设置相关参数，并调用 ai_data_cut 函数切割数据：

```
#3
xlst,ysgn=['AT','V','AP','RH'],'PE'
ftg0='tmp/ccpp_'
ai_data_cut(df,xlst,ysgn,ftg0,True)
```

注意代码当中的 xlst、ysgn 是数据列的名称。

对应的输出结果如下：

```
tmp/ccpp_xtrain.csv
tmp/ccpp_xtest.csv
tmp/ccpp_ytrain.csv
tmp/ccpp_ytest.csv

x_train
         AT      V       AP     RH
2895  29.39  71.14  1010.97  53.88
7813  25.65  78.92  1010.83  86.56
905   10.22  39.64  1010.72  63.05
5192  20.32  44.60  1015.16  36.35
235   17.37  41.23   998.79  68.44

x_test
         AT      V       AP     RH
3854  26.34  69.45  1013.87  54.15
4785  25.06  64.63  1020.66  54.93
6406   8.34  40.96  1023.28  89.45
175   26.69  70.36  1006.82  68.29
7687  17.46  53.29  1018.07  91.01

y_train
2895    430.50
```

```
    7813    434.78
    905     477.22
    5192    460.21
    235     461.08
    Name: PE, dtype: float64

    y_test
    3854    438.50
    4785    446.57
    6406    483.92
    175     435.04
    7687    458.06
    Name: PE, dtype: float64

    ok!
```

ai_data_cut 数据切割函数，是位于 ztop_ai 极宽机器学习模块中的工具函数，为了便于学习，我们将该函数复制到主程序。

数据切割函数

ai_data_cut 数据切割函数代码如下：

```
    def ai_data_cut(df,xlst,ysgn,ftg0,fgPr=False):
        x,y= df[xlst],df[ysgn]
        x_train, x_test, y_train, y_test = train_test_split(x, y, 
random_state=1)
        #

fss=ftg0+'xtrain.csv';x_train.to_csv(fss,index=False);print(fss)

fss=ftg0+'xtest.csv';x_test.to_csv(fss,index=False);print(fss)

fss=ftg0+'ytrain.csv';y_train.to_csv(fss,index=False,header=True)
;print(fss)
```

```
        fss=ftg0+'ytest.csv';y_test.to_csv(fss,index=False,
header=True);print(fss)
        #
        if fgPr:
            print('\nx_train');print(x_train.tail())
            print('\nx_test');print(x_test.tail())
            print('\ny_train');print(y_train.tail())
            print('\ny_test');print(y_test.tail())
```

函数代码很简单，就是调用 Sklearn 模块的 train_test_split 数据分割函数，把原始数据切割为 4 组：

- 训练数据集 x_train，文件名是 ccpp_xtrain.csv。
- 训练数据集对应的答案数据集 y_train，文件名是 ccpp_ytrain.csv。
- 测试数据集 x_test，文件名是 ccpp_xtest.csv。
- 测试数据集对应的答案数据集 y_test，文件名是 ccpp_ytest.csv。

以上数据文件，在案例中输出在 tmp 目录，我们在 dat 目录中也复制了一套，以便于后面的案例学习。

案例 8-3：读取CCPP数据集

案例 8-3 文件名是 zai403_ccpp03.py，介绍如何读取切割后的 CCPP 数据集，下面我们分组进行说明。

第 1 组代码，设置文件名前缀：

```
    #1
    fsr0='dat/ccpp_'
    print('#2,',fsr0)
```

代码很简单，设置文件名前缀，调用 ztop_ai 极宽机器学习模块中的 ai_dat_rd 数据读取函数，读取所需的数据，并保存到对应的各个变量。

需要注意的是，机器学习所需的数据文件的文件名都是统一的前缀，

后面的字母分别代表各种数据，需要预先准备好。

训练数据集 x_train，文件名是 xxx_xtrain.csv。

训练数据集对应的答案数据集 y_train，文件名是 xxx_ytrain.csv。

测试数据集 x_test，文件名是 xxx_xtest.csv。

测试数据集对应的答案数据集 y_test，文件名是 xxx_ytest.csv。

如果调用 ai_data_cut 程序切割数据，会自动生成相应的数据文件。

第 1 组程序对应的文件名在第 2 组输出信息的开头部分，内容如下：

```
dat/ccpp_xtrain.csv
dat/ccpp_xtest.csv
dat/ccpp_ytrain.csv
dat/ccpp_ytest.csv
```

第 2 组、第 3 组代码分别设置 k0=1、k0=10，调用 ztop_ai 极宽机器学习模块中的 ai_dat_rd 数据读取函数，读取所需的数据，并保存到对应的各个变量，代码如下：

```
#2
k0=1
print('\n#2,k0',k0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0,k0,True)

#3
k0=10
print('\n#3,k0',k0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0,k0,True)
```

两组代码的输出数据大部分相同，唯一有差别的是两组 y 字头的答案数据：

- 训练数据集对应的答案数据集 y_train。
- 测试数据集对应的答案数据集 y_test。

对应的输出信息如下：

```
y_train
      PE
7171  430
7172  435
7173  477
7174  460
7175  461

y_test
      PE
2387  438
2388  447
2389  484
2390  435
2391  458
```
第 2 组输出信息，k0=1

```
y_train
      PE
7171  4305
7172  4348
7173  4772
7174  4602
7175  4611

y_test
      PE
2387  4385
2388  4466
2389  4839
2390  4350
2391  4581
```
第 3 组输出信息，k0=10

由输出信息可以看出，第 3 组的输出数据大约是第 2 组的 10 倍，具体原因，我们参看 ai_dat_rd 数据读取函数源码。

数据读取函数

案例 8-3 中调用的 zai.ai_dat_rd 数据读取函数，代码如下：

```
def ai_dat_rd(fsr0,k0=1,fgPr=False):
    #1
    fss=fsr0+'xtrain.csv';x_train=pd.read_csv(fss,index_col=
False);print(fss)
    fss=fsr0+'xtest.csv';x_test=pd.read_csv(fss,index_col=
False);print(fss)
    fss=fsr0+'ytrain.csv';y_train=pd.read_csv(fss,index_col=
False);print(fss)
    fss=fsr0+'ytest.csv';y_test=pd.read_csv(fss,index_col=
False);print(fss)
    #2
```

```
    ysgn=y_train.columns[0];#print('y',ysgn)
    y_train[ysgn]=round(y_train[ysgn]*k0).astype(int)
    y_test[ysgn]=round(y_test[ysgn]*k0).astype(int)
    #3
    if fgPr:
        print('\nx_train');print(x_train.tail())
        print('\nx_test');print(x_test.tail())
        print('\ny_train');print(y_train.tail())
        print('\ny_test');print(y_test.tail())
    #4
    return  x_train, x_test, y_train, y_test
```

函数很简单，我们按代码组分别进行说明。

- 第 1 组代码：读取各个数据文件，并保存到相应的变量。
- 第 2 组代码：需要注意，把 y 字头的答案数据改为整数类型，个别原始数据值较小，需要乘以 k0，进行数值放大，这是因为部分机器学习算法的答案数据只支持整数格式，很多初学者都在此处遇到瓶颈。
- 第 3 组代码：根据 fgPr 布尔变量，输出相关的变量数据信息。
- 第 4 组代码：需要注意，函数的返回值是多组，注意好各个变量的名称和次序，一般是直接复制到调用主程序。

8.2 机器学习统一接口函数

在前面的案例中，我们介绍的各种机器学习算法都位于 ztop_ai 极宽机器学习模块中，而且函数名称都是 mx_xxx 格式，调用的数据接口也是一致的。

这样的二次封装，虽然大大简化了机器学习算法的调用过程，降低了学习难度，但作为机器学习的组合调用还不够，需要再追加一个标准的机器学习调用接口函数。

案例 8-4：机器学习统一接口

案例 8-4 文件名是 zai404_mx_01.py，介绍如何使用标准的接口，统一调用机器学习算法函数建模、分析以及得出预测结果，下面我们分组逐一进行介绍。

第 1 组代码，设置有关参数，并读取已经分割好的训练数据、测试数据和对应的答案数据，其中 xlst、ysgn 是数据源的数据字段名称，相关代码如下：

```
#1
fsr0='dat/ccpp_'
print('#1',fsr0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
```

对应的输出信息是：

```
#1 dat/ccpp_
dat/ccpp_xtrain.csv
dat/ccpp_xtest.csv
dat/ccpp_ytrain.csv
dat/ccpp_ytest.csv
```

输出信息包括文件名前缀，以及各组数据的文件名。

第 2 组代码，设置机器学习函数名称，并调用统一的接口函数 mx_fun010，相关代码如下：

```
#2
print('\n#2,mx_line')
#mx_fun=zai.mx_line
funSgn='line'
tim0=arrow.now()
dacc,df9=mx_fun010(funSgn,x_train, x_test, y_train, y_test,
5,False)
tn=zt.timNSec('',tim0,True)
```

第 2 组代码设置的机器学习函数是线性回归函数，并计算运行时间，其中函数名称是通过 funSgn 字符串代码传递的。

对应的结果数据如下：

```
#2,mx_line
@mx:mx_sum,kok:99.96%
0.01 s, 09:11:38 ,t0, 09:11:38
```

第 3 组代码与第 2 组代码类似，但采用的是逻辑回归算法，此外，调用时最后一个输入参数 fgDebug=True，采用调试模式输出更多信息，相关代码如下：

```
#3
print('\n#3,mx_log')
funSgn='log'
tim0=arrow.now()
dacc,df9=mx_fun010(funSgn,x_train, x_test, y_train, y_test,
5,False,True)
tn=zt.timNSec('',tim0,True)
```

对应的输出信息是：

```
#3,mx_log
ai_acc_xed
      AT      V       AP       RH      y_test   y_pred   ysub
ysub2     y_test_div      ysubk
0   17.80   43.72   1008.71   78.50   459      466      -7
7         459.0           1.525054
1   29.60   71.14   1011.46   52.69   431      436      -5
5         431.0           1.160093
2   11.06   36.71   1021.67   80.44   474      474      0
0         474.0           0.000000
3   30.06   67.25   1017.63   53.59   435      440      -5
5         435.0           1.149425
4   19.88   47.03   1012.27   91.99   456      466      -10
10        456.0           2.192982
```

```
test,2392,npred,2392,dsum,2384
acc-kok: 99.67%, MAE:5.04, MSE:42.68, RMSE:6.53
@fun name: mx_log
@mx:mx_sum,kok:99.67%
0.99 s, 09:11:39 ,t0, 09:11:38
```

在调试模式下，会输出较多信息。

在最后的结果中：

- 逻辑回归算法的准确度是 99.67%，耗时 1 秒，但因为是调试模式，输出信息过多，所以无法反映真实的运行时间。
- 线性回归算法的准确度是 99.96%，耗时 0.01 秒。

1. 统一接口函数

在案例中使用的统一的接口函数 zai.mx_fun010，原本位于 ztop_ai 极宽机器学习模块中，为了大家学习方便，我们复制了一份到主程序。

下面对 zai.mx_fun010 函数分组进行讲解。

zai.mx_fun010 函数的接口定义如下：

```
    def mx_fun010(funSgn,x_train, x_test, y_train, y_test,yk0=5,
fgInt=False,fgDebug=False):
```

其中各函数定义如下。

- funSgn：字符串格式，机器学习函数名称代码，实际调用的函数是 ztop_ai 极宽机器学习模块中二次封装的函数，函数名一般是 mx_xxx。
- x_train,x_test,y_train,y_test：学习和测试数据集，使用 Pandas 的 Dataframe 格式。
- yk0=5：结果数据误差 k 值，默认是 5，表示 5%；整数模式设置为 1。
- fgInt=False：整数结果模式，默认是 False。
- fgDebug=False：调试模式，默认是 False。

第 1 组代码，设置结果变量 df9，机器学习函数 mx_fun，并且调用该函数进行机器学习，构建算法模型，并保存到变量模型。

```
#1
df9=x_test.copy()
mx_fun=mxfunSgn[funSgn]
mx =mx_fun(x_train.values,y_train.values)
```

第 2 组代码，调用 mx 模型变量内置的 Predict 预测函数，生成预测结果并保存到变量 y_pred，df9 的相关参数：

```
#2
y_pred = mx.predict(x_test.values)
df9['y_test'],df9['y_pred']=y_test,y_pred
```

第 3 组代码，默认是非整数模式，如果 fgInt 结果数据为整数模式才执行，则需要把结果变量 df9 中的数据格式转换为整数格式。

Iris 爱丽丝数据集和一些分类算法，其结果都需要采用整数模式，CCPP 数据集的案例，输入时需要对结果数据进行整数化处理，但结果无须采用整数格式。

```
#3
if fgInt:
    df9['y_predsr']=df9['y_pred']
    df9['y_pred']=round(df9['y_predsr']) .astype(int)
```

第 4 组代码，调用自定义的 ai_acc_xed 效果评估函数，生成结果数据，注意 fgDebug 调试模式参数，也会传递至 ai_acc_xed 效果评估函数。

```
#4
dacc=ai_acc_xed(df9,yk0,fgDebug)
```

第 5 组代码，调试模式变量 fgDebug=True，输出结果数据变量 df9 的尾部数据，并把 df9 保存到文件 tmp/df9_pred.csv。

```
#5
if fgDebug:
```

```
        #print(df9.head())
        print('@fun name:',mx_fun.__name__)
        df9.to_csv('tmp/df9_pred.csv');
```

第 6 组代码，输出评估结果，返回 dacc 评估数值和结果变量 df9，其中 dacc 评估数值，采用的是百分比数据。

```
    #6
    print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
    return dacc,df9
```

2．机器学习算法名称代码

在 zai.mx_fun010 函数中使用的 funSgn 字符串变量，是机器学习函数的代码名称，定义在 ztop_ai 极宽机器学习模块中，位置在 mx_xxx 函数定义之后，mx_fun010 函数定义前的相关代码如下：

```
    #--------mx.fun.sgn

    mxfunLst=['line','log','bayes','knn','forest','dtree','gbdt',
'svm','svmcr','mlp','mlpreg']
    mxfunSgn={'line':mx_line,
            'log':mx_log,
            'bayes':mx_bayes,
            'knn':mx_knn,
            'forest':mx_forest,
            'dtree':mx_dtree,
            'gbdt':mx_GBDT,
            'svm':mx_svm,
            'svmcr':mx_svm_cross,
            'mlp':mx_MLP,
            'mlpreg':mx_MLP_reg
            }
```

其中：

- mxfunLst 是列表格式，各个机器学习算法函数的缩写代码。
- mxfunSgn 是字典格式，各个机器学习算法函数的缩写代码和对应的函数名称。

mxfunLst 和 mxfunSgn 没有按一般的参数定义放在模块前部，而是放在模块中部这个位置，是因为字典中的函数名称必须先定义才能设置，不然会出现错误。

mxfunLst 函数列表和 mxfunSgn 字典变量，随着 ztop_ai 极宽机器学习模块的升级，会不断进行优化调整。

3. 效果评估函数

ai_acc_xed 效果评估函数位于 ztop_ai 极宽机器学习模块中，主要用于评估机器学习算法函数的效果，下面我们分组逐一进行讲解。

ai_acc_xed 效果评估函数的接口定义如下：

```
def ai_acc_xed(df9,ky0=5,fgDebug=True):
```

其中：

- df9，Pandas 的 Dataframe 格式，结果数据保存变量
- ky0，结果数据误差 k 值，默认是 5，表示 5%；整数模式设置为 1。
- fgDebug，调试模式变量，默认为 False。

第 1 组代码，设置预测数据与实际结果数据的绝对误差值，并保存到新增数据列 ysub2：

```
#1
ny_test,ny_pred=len(df9['y_test']),len(df9['y_pred'])
df9['ysub']=df9['y_test']-df9['y_pred']
df9['ysub2']=np.abs(df9['ysub'])
```

第 2 组代码如下：

```
#2
df9['y_test_div']=df9['y_test']
df9.loc[df9['y_test'] == 0, 'y_test_div'] =0.00001
df9['ysubk']=(df9['ysub2']/df9['y_test_div'])*100
dfk=df9[df9['ysubk']<ky0]
dsum=len(dfk['y_pred'])
dacc=dsum/ny_test*100
```

运行流程如下：

- 计算结果数据绝对差值 ysub2 与实际结果数据 y_test 之间的差距，结果为百分比，并保存在数据列 ysubk。
- 提取 ysubk 小于设定的误差值 ky0 的数据到变量 dfk。
- 计算变量 dfk 的长度，结果就是正确的预测结果数目，并保存在变量 dsum。
- 计算 dsum 与测试数据总数目 ny_test 的百分比，结果就是预测结果的准确度，并保存在变量 dacc。

第 3 组代码，如果调试模式变量 fgDebug=True，输出相关数据，并调用 Sklearn 的测试模块 Metrics，进行进一步的测试：

```
#3
    if fgDebug:
        print('\nai_acc_xed')
        print(df9.head())
        y_test,y_pred=df9['y_test'],df9['y_pred']
        print('\ntest,{0},npred,{1},dsum,{2}'.format(ny_test,
ny_pred,dsum))
        dmae=metrics.mean_absolute_error(y_test, y_pred)
        dmse=metrics.mean_squared_error(y_test, y_pred)
        drmse=np.sqrt(metrics.mean_squared_error(y_test,
y_pred))
        print('acc-kok:  {0:.2f}%,  MAE:{1:.2f},  MSE:{2:.2f},
RMSE:{3:.2f}'.format(dacc,dmae,dmse,drmse))
```

其中对应的输出结果如下：

```
    ai_acc_xed
         AT        V         AP      RH  y_test  y_pred  ysub  ysub2
y_test_div    ysubk
    0  17.80  43.72  1008.71  78.50     459     466    -7      7
459.0  1.525054
    1  29.60  71.14  1011.46  52.69     431     436    -5      5
431.0  1.160093
    2  11.06  36.71  1021.67  80.44     474     474     0      0
474.0  0.000000
    3  30.06  67.25  1017.63  53.59     435     440    -5      5
435.0  1.149425
    4  19.88  47.03  1012.27  91.99     456     466   -10     10
456.0  2.192982

    test,2392,npred,2392,dsum,1283
    acc-kok: 53.64%, MAE:5.04, MSE:42.68, RMSE:6.53
```

4. 常用评测指标

ai_acc_xed 效果评估函数中的 MAE、MSE、RMSE 都是统计术语，用于结果评测，定义分别如下。

- MAE：Mean Absolute Error，平均绝对误差。
- MSE：Mean Squared Error，均方差、方差，结果数据越接近于 0，说明模型选择和拟合更好，数据预测也越成功。
- RMSE：Root Mean squared Error，均方根、标准差，结果数据越接近于 0，说明模型选择和拟合更好，数据预测也越成功。

还有以下类似术语。

- SSE（The Sum of Squares Due to Error）：和方差、误差平方和。SSE 越接近于 0，说明模型选择和拟合更好，数据预测也越成功。MSE 和 RMSE 因为和 SSE 是同出一宗，所以效果一样。

- R-square（Coefficient of determination）：确定系数，通过数据的变化来表征一个拟合算法的好坏，正常取值范围为[0 1]，越接近 1，表明方程的变量对 y 的解释能力越强，算法模型对数据拟合也较好。
- Adjusted R-square（Degree-of-freedom Adjusted Coefficient of Determination）：修正确定系数。
- Precision：又称 P 指标，表示准确率，即检索出来的条目（比如文档、网页等）有多少是准确的。
- Recal：又称 R 指标，表示召回率，即所有准确的条目有多少被检索出来了。
- F 值（F-Measure）：又称为 F-Score，F 值是正确率和召回率的调和平均值；F 值=正确率×召回率×2 / (正确率+召回率)，F 值即正确率和召回率的调和平均值。
- E 值：表示查准率 P 和查全率 R 的加权平均值，当其中一个为 0 时，E 值为 1。
- AP（Average Precision）：平均正确率，表示不同查全率的点上的正确率的平均值。

案例 8-5：批量调用机器学习算法

在前面的案例中，我们介绍了各种机器学习算法，以及统一的接口，本节我们将通过机器学习算法统一接口，对各种机器学习算法批量调用，同时测试一下各种算法、模型的准确度和运行速度。

案例 8-5 文件名是 zai405_mx_02.py，介绍如何批量调用各种机器学习算法。

第 1 组代码，主要设置各种参数，读取训练和测试数据集。

```
#1
mxlst1=['line','knn','bayes']
mxlst2=['line','log','bayes','knn','forest','dtree','svm','ml
```

```
p','mlpreg']
    #'gbdt','svmcr'

    fsr0='dat/ccpp_'
    print('#1',fsr0)
    x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
```

其中 mxlst1、mxlst2 是批量调用的机器学习字符串代码列表，都是小写格式。

这些标识符源自 ztop_ai 极宽机器学习模块中的 mxfunLst：

```
    mxfunLst=['line','log','bayes','knn','forest','dtree','gbdt',
'svm','svmcr','mlp','mlpreg']
```

随着 ztop_ai 版本的升级，具体代码和标识符可能会有所变动，以最新发布的软件代码为准。

注意 mxlst1、mxlst2 列表中各种标识符的位置、次序只是影响对应机器学习函数的调用次序，对其他结果没有影响。

在 mxlst2 列表中没有 gbdt、svmcr 两种算法的标识符，这是因为测试时这两种算法出现错误，可能是 CCPP 数据不适合这两种算法。这也说明，各种算法模型都有一定的局限性。

第 2 组代码，采用 mxlst1，调用 zai.mx_funlst 批量调用函数：

```
    #2
    print('\n#2,mxlst1')
    zai.mx_funlst(mxlst1,x_train, x_test, y_train, y_test)
```

对应的输出信息是：

```
    #2,mxlst1


     line
    @mx:mx_sum,kok:99.96%
    0.01 s, 09:18:05 ,t0, 09:18:05
```

```
 knn
@mx:mx_sum,kok:99.50%
0.04 s, 09:18:05 ,t0, 09:18:05

 bayes
@mx:mx_sum,kok:99.83%
0.02 s, 09:18:05 ,t0, 09:18:05
```

mxlst1 只有 Line（线性回归）、KNN（k 近邻）、Bayes（贝叶斯）3 种算法，函数运行速度都很快，在 0.01~0.04 秒左右，准确度也差不多，都在 99%左右。

第 3 组代码，采用 mxlst2，调用 zai.mx_funlst 批量调用函数：

```
#3
print('\n#3,mxlst2')
zai.mx_funlst(mxlst2,x_train, x_test, y_train, y_test,False)
```

对应的输出信息是：

```
#3,mxlst2

 line
@mx:mx_sum,kok:99.96%
0.01 s, 09:18:05 ,t0, 09:18:05

 log
@mx:mx_sum,kok:99.67%
0.96 s, 09:18:06 ,t0, 09:18:05

 bayes
@mx:mx_sum,kok:99.83%
0.02 s, 09:18:06 ,t0, 09:18:06

 knn
```

```
@mx:mx_sum,kok:99.50%
0.04 s, 09:18:06 ,t0, 09:18:06

 forest
@mx:mx_sum,kok:99.67%
0.27 s, 09:18:06 ,t0, 09:18:06

 dtree
@mx:mx_sum,kok:99.75%
0.07 s, 09:18:06 ,t0, 09:18:06

 svm
@mx:mx_sum,kok:96.40%
29.65 s, 09:18:36 ,t0, 09:18:06

 mlp
@mx:mx_sum,kok:99.75%
1.42 s, 09:18:37 ,t0, 09:18:36

 mlpreg
  y = column_or_1d(y, warn=True)
@mx:mx_sum,kok:99.96%
1.22 s, 09:18:38 ,t0, 09:18:37
```

在输出信息中，SVM 向量机算法所需时间最久，将近 30 秒；MLP 和 Mlpreg 神经网络算法，以及 Log 回归算法的时间也较长，大约 1～1.4 秒，其他的机器学习算法一般在 1 秒之内。

在准确度方面：

- 除了 SVM 向量机算法准确度是 96.4%，其他的都在 99%左右。
- SVM 向量机算法较低，但并不代表 SVM 向量机算法效果最差，只是说明该算法不太适合 CCPP 数据集。
- 不少机器学习算法内部都采用了随机种子，所以每次运行的结果可能会有略微差异，但整体数值都差不多。

批量调用算法函数接口

在案例中，批量调用机器学习的算法，使用的是 zai.mx_funlst 批量调用函数，相关代码如下：

```
    def mx_funlst(funlst,x_train, x_test, y_train, y_test,yk0=5,
fgInt=False):
        for funsgn in funlst:
            print('\n',funsgn)
            tim0=arrow.now()
            mx_fun010(funsgn,x_train, x_test, y_train, y_test,yk0,
fgInt)
            zt.timNSec('',tim0,True)
```

由以上代码可以看出，有了统一调用接口函数 mx_fun010 后，批量调用机器学习算法其实非常简单，一个 for 循环即可完成。

案例 8-6：一体化调用

下面我们介绍一个一体化的机器学习调用案例，从原始数据到直接分割数据，然后进行学习训练和结果预测，更加贴近实盘的调用模式。

案例 8-6 文件名是 zai406_mx_03.py，演示如何一体化调用机器学习算法，程序很简单，核心代码如下：

```
    #1
    xlst,ysgn=['AT','V','AP','RH'],'PE'
    df=pd.read_csv('dat/ccpp.csv',index_col=False)

    #2
    print('\n#2,mx_line')
    funsgn='line'
    tim0=arrow.now()
    zai.mx_fun_call(df,xlst,ysgn,funsgn)
    tn=zt.timNSec('',tim0,True)
```

从以上代码可以看出，一体化调用代码更加简洁，因为省略了各个切割好的训练数据，以及读取数据的参数设置和加载。

核心代码只有一条，调用 mx_fun_call 函数：

```
zai.mx_fun_call(df,xlst,ysgn,funsgn)
```

可以看出，一体化调用的函数定义接口也简化了很多。

程序运行结果如下：

```
#2,mx_line

n_tran:, 7176  ,ntst, 2392  ,dimension:, 4 ,kbin, False

y_pred,预测
@mx:mx_sum,kok:99.96%
0.02 s, 10:05:23 ,t0, 10:05:23
```

一体化调用函数

一体化调用函数 mx_fun_call 位于 ztop_ai 极宽机器学习模块中，下面我们分组进行讲解。

函数定义与接口，代码如下：

```
def mx_fun_call(df,xlst,ysgn,funSgn,yksiz=1,yk0=5,fgInt=False,
fgDebug=False):
```

其中各字母含义如下。

- df：数据源，可以是原始数据，也可以是归一化处理的衍生数据，不过必须全部是数字。
- xlst：参数数据集字段名称，源自 df 的字段名。
- ysgn：对应的答案字段名称。
- funSgn：机器学习算法字符串代码。
- yksiz：结果数据缩放比例，默认是 1。

- ky0：结果数据误差 k 值，默认是 5，表示 5%；整数模式设置为 1。
- fgInt：整数结果模式变量，默认为 False。
- fgDebug：调试模式变量，默认为 False。

第 1 组代码，设置 ysgn 答案数据列的数据格式为整数，如果 ysgn 原数据过小，需要设置合适的 yksiz 参数进行调整：

```
#1
df[ysgn]=df[ysgn].astype(float)
df[ysgn]=round(df[ysgn]*yksiz).astype(int)
```

第 2 组代码，设置主数据集 x 和对应的答案数据集 y：

```
#2
x,y= df[xlst],df[ysgn]
```

第 3 组代码，分割数据为 x_train, x_test, y_train, y_test：

```
#3
x_train, x_test, y_train, y_test = train_test_split(x, y, random_
state=1)
num_train, num_feat = x_train.shape
num_test, num_feat = x_test.shape
print('\nn_tran:,',num_train,' ,ntst,' , num_test, ' ,dimension:,
',num_feat,',kbin,',fgInt)
```

第 4 组代码，根据 funSgn 机器学习算法标识符，设置对应的函数，并调用该函数进行建模，结果保存在变量 mx：

```
#4
print('\ny_pred,预测')
df9=x_test.copy()
mx_fun=mxfunSgn[funSgn]
mx =mx_fun(x_train.values,y_train.values)
```

第 5 组代码，调用模型变量 mx 的内置 Predict 预测函数，生成预测数据 y_pred，并保存到结果变量 df9：

```
#5
y_pred = mx.predict(x_test.values)
df9['y_test'],df9['y_pred']=y_test,y_pred
```

第 6 组代码，如果整数答案模式 fgInt=True，则把结果变量 df9 中的 y_pred 数据列改为整数格式：

```
#6
if fgInt:
    df9['y_predsr']=df9['y_pred']
    df9['y_pred']=round(df9['y_predsr']) .astype(int)
```

第 7 组代码，调用 ai_acc_xed 效果评测函数，计算预测结果的准确度：

```
#7
dacc=ai_acc_xed(df9,yk0,fgDebug)
```

第 8 组代码，如果调试模式 fgDebug =True，则输出结果变量 df9 的部分数据，并保存到文件 tmp/df9_pred.csv：

```
#8
if fgDebug:
    #print(df9.head())
    print('@fun name:',mx_fun.__name__)
    df.to_csv('tmp/df_sr.csv');
    df9.to_csv('tmp/df9_pred.csv');
```

第 9 组代码，输出准确度数据，返回结果数据 dacc,df9：

```
#9
print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
return dacc,df9
```

8.3 模型预制与保存

在机器学习算法批量调用案例中，SVM 向量机算法所需时间最久，将

近 30 秒，MLP 和 Mlpreg 神经网络算法，以及 Log 回归算法的时间也较长，大约 1～1.4 秒。

虽然这几种算法的运算时间，都在 1～1.4 秒之间，但相对于 SVM 向量机算法的 30 秒，算是比较快的了。

CCPP 数据集只有 1 万条数据，在实盘当中，单只股票每天的 ticks 数据就超过一万条，在足彩数据中，GID 基本比赛数据有 7 万条，总的赔率数据有 200 多万条。

在实盘中，往往耗时最多的是机器学习构建算法模型，而真正预测的数据一般并不多，也无须很长的计算时间。

建模使用的是历史数据，经常是数万条记录，甚至百万、千万级数据，对于某些大数据项目更是以亿单位为起点。

实盘预测往往只是采用实时数据，一般都是当天的最新数据，配合一些其他数据，比如均线数据，所以数据量并不大。

以往，我们的案例都是把建模和预测放在一起，如果能够分开，实盘分析无须建模，直接预测，速度可以至少快 10 倍以上。

Python 语言的 Pickle 模块，支持二进制的数据和程序一体化储存，是非常理想的机器学习模型储存工具。

但在 Sklearn 模块中，内置了高效的模型持久化模块 Joblib，将模型保存至硬盘，并且从硬盘加载，效率更高。

使用时，代码也非常简单：

```
from sklearn.externals import joblib
#假设 mx 是一种机器学习的模型
joblib.dump(mx, 'mx.pkl')
mx = joblib.load('mx. pkl')
```

Joblib 使用文件格式是 pkl，本质上也是 Pickle 模块的一种文件保存格式，而且，Pickle 也可以直接读取 pkl 文件。

案例 8-7：储存算法模型

案例 8-7 文件名是 zai407_mx_04.py，介绍如何使用 Sklearn 模块中的 Joblib 函数，储存并读取机器学习算法模型，下面我们分组逐一进行介绍。

第 1 组代码，设置参数，读取训练和测试数据集，以及 mx 模型文件名。

```
#1
fsr0='dat/ccpp_'
print('#1',fsr0)
xlst,ysgn=['AT','V','AP','RH'],'PE'
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
funSgn,ftg='svm','tmp/ccpp_svm.pkl'
```

第 2 组代码，通过 funSgn 调用 zai.ai_f_mxWr 函数，对输入数据运行机器学习算法，并保存到文件：

```
#2
print('\n#2,mx_svm.wr')
tim0=arrow.now()
zai.ai_f_mxWr(ftg,funSgn,x_train, y_train)
tn=zt.timNSec('',tim0,True)
```

对应的输出信息是：

```
#2,mx_svm.wr
27.88 s, 11:09:44 ,t0, 11:09:16
```

在案例中使用的是 SVM 向量机模型，该模型耗时较久，以前的案例调用大约是 30 秒，此处，训练时间大约是 27.88 秒。

第 3 组代码，读取 mx 模型数据：

```
#3
tim0=arrow.now()
print('\n#3,mx_svm.rd')
mx = joblib.load(ftg)
tn=zt.timNSec('',tim0,True)
```

对应的输出信息是：

```
#3,mx_svm.rd
0.01 s, 11:09:44 ,t0, 11:09:44
```

读取数据很快，才 0.01 秒。

第 4 组代码，使用读取的 mx 模型数据，进行预测分析：

```
#4
print('\n#4,mx_svm')
tim0=arrow.now()
zai.mx_fun8mx(mx,x_test,y_test)
tn=zt.timNSec('',tim0,True)
```

对应的输出信息是：

```
#4,mx_svm
@mx:mx_sum,kok:96.40%
1.83 s, 11:09:46 ,t0, 11:09:44
```

准确度是 96.4%，运行时间是 1.83 秒，比以前的 30 秒快了近 20 倍。

1. 模型保存函数

模型保存函数 ai_f_mxWr 位于 ztop_ai 极宽机器学习模块中，代码如下：

```
def ai_f_mxWr(ftg,funSgn,x_train, y_train):
    #1
    mx_fun=mxfunSgn[funSgn]
    mx =mx_fun(x_train.values,y_train.values)
    #2
    joblib.dump(mx,ftg)
```

以上代码非常简单，大家自己解读即可。

2. 模型预测函数

模型预测函数 mx_fun8mx 通过预先保存或者生成的算法模型分析输入

数据，函数位于 ztop_ai 极宽机器学习模块中，代码如下：

```
def mx_fun8mx(mx,x_test,y_test,yk0=5,fgInt=False,fgDebug=False):
    #1
    df9=x_test.copy()
    #mx=....
    #2
    y_pred = mx.predict(x_test.values)
    df9['y_test'],df9['y_pred']=y_test,y_pred
    #3
    if fgInt:
        df9['y_predsr']=df9['y_pred']
        df9['y_pred']=round(df9['y_predsr']).astype(int)

    #4
    dacc=ai_acc_xed(df9,yk0,fgDebug)
    #5
    if fgDebug:
        #print(df9.head())
        print('@fun name:',mx_fun.__name__)
        df9.to_csv('tmp/df9_pred.csv');
    #
    #6
    print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
    return dacc,df9
```

以上函数代码看起来较长，但其实不过是 mx_fun010 函数后面的一部分，大家参考函数 mx_fun010 的介绍即可理解以上代码。

事实上，模型保存函数 ai_f_mxWr 和模型预测函数 mx_fun8mx，都是从 mx_fun010 函数中分割出来的，有兴趣的读者可以对比分析一下。

案例 8-8：批量储存算法模型

需要说明的是，Sklearn 模块保存的机器学习算法模型，不同的算法模型保存的模型不同，即使是同一个算法，不同的训练数据，保存的模型也不同。

例如 CCPP 数据集，如果每次调用数据分割函数，动态生成训练数据和测试数据，那么每次的训练模型都要采用不同的文件名进行保存，不然会出现混淆。

为了提高效率，我们需要批量储存算法模型，案例 8-8 就是介绍如何批量储存不同的算法模型。

案例 8-8 文件名是 zai408_mx_05.py，程序很简单，核心代码如下：

```
#1
fsr0='dat/ccpp_'
print('#1',fsr0)
mxlst=['line','log','bayes','knn']
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
ftg0='tmp/ccpp_'

#2
print('\n#2,mx_svm.wr')
zai.ai_f_mxWrlst(ftg0,mxlst,x_train, y_train)
```

以上程序除了最后一行代码，其他的我们都已经介绍过：

```
zai.ai_f_mxWrlst(ftg0,mxlst,x_train, y_train)
```

这一行代码是本案例的核心，调用 ztop_ai 极宽机器学习模块中的 ai_f_mxWrlst 函数，批量储存 CCPP 数据集的相关模型代码，案例中使用了 Line（线性回归）、Log（逻辑回归）、Bayes（贝叶斯）、kNN（k 近邻）4 种不同的机器学习算法。

对应的输出信息如下：

```
#2,mx_svm.wr

 tmp/ccpp_line.pkl
0.0 s, 11:36:22 ,t0, 11:36:22

 tmp/ccpp_log.pkl
0.97 s, 11:36:23 ,t0, 11:36:22

 tmp/ccpp_bayes.pkl
0.01 s, 11:36:23 ,t0, 11:36:23

 tmp/ccpp_knn.pkl
0.01s, 11:36:23 ,t0, 11:36:23
```

几种算法的时间都差不多，除了 Log 逻辑回归算法用了 1 秒，其他基本上都是零点零几秒，除了以上输出信息，程序还在 tmp 目录下生产了多个对应的 pkl 算法模型文件，如图 8-1 所示。

E:\zwPython\zc_demo\tmp

名称	大小	修改日期	类型
ccpp_bayes.pkl	7 KB	2017/2/19 11:36	PKL 文件
ccpp_knn.pkl	559 KB	2017/2/19 11:36	PKL 文件
ccpp_line.pkl	1 KB	2017/2/19 11:36	PKL 文件
ccpp_log.pkl	5 KB	2017/2/19 11:36	PKL 文件
ccpp_svm.pkl	8,519 KB	2017/2/19 11:09	PKL 文件

图 8-1 机器学习算法模型文件

由图 8-1 可以看出，各种算法模型的文件大小差别很大，SVM 向量机的模型文件有 8MB 多，Line 线性回归算法的模型文件只有 1KB 大小。

需要的注意的是，不同规模的训练数据，其模型文件的大小也是不同的，后面的实盘案例，我们会继续进一步介绍。

批量模型储存函数

ai_f_mxWrlst 批量储存函数位于 ztop_ai 极宽机器学习模块中，函数代码如下：

```
def ai_f_mxWrlst(ftg0,funlst,x_train, y_train):
    for funSgn in funlst:
        ftg=ftg0+funSgn+'.pkl'
        print('\n',ftg)
        tim0=arrow.now()
        ai_f_mxWr(ftg,funSgn,x_train, y_train)
        zt.timNSec('',tim0,True)
```

以上函数代码很简单，与 mx_funlst 批量调用函数类似，大家可以对照进行解读。

案例 8-9：批量加载算法模型

做生意讲究的是有进有出，同样，我们储存算法模型的最终目的也是为了实盘应用。

前面我们介绍了批量储存算法模型的案例，下面我们介绍批量读取（加载）储存算法模型。

原以为批量读取算法模型，比批量储存算法模型简单，经过实际案例测试才发现，批量读取比批量储存更加复杂。

其原因在于，储存只要把数据保存到磁盘就行了，而批量读取数据时，读取后的数据，会直接进入程序的数据处理流程当中，如果处理不当，不但不会提升整体性能，反而会干扰其他数据，造成额外错误。

案例 8-9 文件名是 zai409_mx_06.py，介绍如何批量加载保存的机器学习算法模型，并进行测试，下面我们分组对代码进行讲解。

第 1 组代码，设置参数，读取测试数据：

```
#1
fsr0='dat/ccpp_'
print('#1',fsr0)
mxlst=['line','log','bayes','knn','forest','dtree','svm','mlp
```

```
',' mlpreg']
    x_test=zai.ai_f_datRd010(fsr0+'xtest.csv')
    y_test=zai.ai_f_datRd010(fsr0+'ytest.csv',1)
```

因为有存储的算法模型，我们无须读取训练数据，只需要读取测试数据和对应的结果数据即可，所以这里没有使用 ai_dat_rd 多组数据读取函数，调用的是 zai.ai_f_datRd010 单组数据读取函数，函数代码如下：

```
def ai_f_datRd010(fsr,k0=0,fgPr=False):
    #1
    df=pd.read_csv(fsr,index_col=False);
    #2
    if k0>0:
        ysgn=df.columns[0];#print('y',ysgn)
        df[ysgn]=round(df[ysgn]*k0).astype(int)

    #3
    if fgPr:
        print('\n',fsr);print(df.tail())

    #4
    return  df
```

在 ai_f_datRd010 单组数据读取函数中，如果读取主数据，则参数 k0=0，如果读取以 y 开头的答案数据，则需要进行缩放，并且转换为整数格式。

所以在第 1 组代码中，x_test 主数据和结果数据 y_test 的调用参数是不同的。

此外，为了保持函数格式的一致性，我们把 ai_dat_rd 多组数据读取函数统一归纳在 ai_f_xxx 文件函数中，将函数名称改为：

```
def ai_f_datRd(fsr0,k0=1,fgPr=False):
```

但为了兼容以往的案例，原 ai_dat_rd 函数依然保留，在以后的版本中会进行优化处理。

第 2 组代码，批量读取算法模型：

```
#2
zai.xmodel={}
print('\n#2,ai_f_mxRdlst')
zai.ai_f_mxRdlst(fsr0,mxlst)
```

调用的是 zai.ai_f_mxRdlst 函数批量读取各个算法模型文件，对应的函数代码如下：

```
def ai_f_mxRdlst(fsr0,funlst):
    for funSgn in funlst:
        fss=fsr0+funSgn+'.pkl'
        print(fss)
        xmodel[funSgn]=joblib.load(fss)
```

读取其实很简单，难点在于，读取后如何处理数据。

为了提高性能，我们在 ztop_ai 极宽机器学习模块当中，设置了一个字典格式的全局变量 zai.xmodel，并且在批量调用前，进行清零处理。

第 3 组代码，批量调用算法模型，并进行分析预测：

```
#3
print('\n#3,mx_funlst8mx')
zai.mx_funlst8mx(mxlst, x_test,  y_test,yk0=5,fgInt=False)
```

实际分析预测调用的函数是 zai.mx_funlst8mx，对应的代码如下：

```
def mx_funlst8mx(mxlst, x_test,  y_test,yk0=5,fgInt=False):
    for msgn in mxlst:
        print('@msgn:',msgn)
        tim0=arrow.now()
        mx=xmodel[msgn]
        mx_fun8mx(mx, x_test, y_test,yk0,fgInt)
        zt.timNSec('',tim0,True)
```

zai.mx_funlst8mx 函数代码很简单，关键就在于：

```
        mx=xmodel[msgn]
```

使用 xmodel 中保存的模型，并进行分析预测。

变量 yk0 是误差精度的 k 阈值，通常，我们使用的都是 yk0=5，即结果误差在 5%以内，下面我们把第 3 组调用代码的参数设置为 yk0=1，按 1%的误差精度，再进行一次测试。

第 3 组代码，分析的结果是：

```
#3,mx_funlst8mx
@msgn: line
@mx:mx_sum,kok:99.96%
0.01 s, 13:10:01 ,t0, 13:10:01
@msgn: log
@mx:mx_sum,kok:99.67%
0.01 s, 13:10:01 ,t0, 13:10:01
@msgn: bayes
@mx:mx_sum,kok:99.83%
0.01 s, 13:10:01 ,t0, 13:10:01
@msgn: knn
@mx:mx_sum,kok:99.50%
0.03 s, 13:10:01 ,t0, 13:10:01
@msgn: forest
@mx:mx_sum,kok:99.67%
0.02 s, 13:10:01 ,t0, 13:10:01
@msgn: dtree
@mx:mx_sum,kok:99.75%
0.01 s, 13:10:01 ,t0, 13:10:01
@msgn: svm
@mx:mx_sum,kok:96.40%
1.84 s, 13:10:03 ,t0, 13:10:01
@msgn: mlp
@mx:mx_sum,kok:99.83%
```

```
#3,mx_funlst8mx
@msgn: line
@mx:mx_sum,kok:68.52%
0.01 s, 13:16:34 ,t0, 13:16:34
@msgn: log
@mx:mx_sum,kok:53.64%
0.01 s, 13:16:34 ,t0, 13:16:34
@msgn: bayes
@mx:mx_sum,kok:54.72%
0.01 s, 13:16:34 ,t0, 13:16:34
@msgn: knn
@mx:mx_sum,kok:66.14%
0.03 s, 13:16:34 ,t0, 13:16:34
@msgn: forest
@mx:mx_sum,kok:78.64%
0.02 s, 13:16:34 ,t0, 13:16:34
@msgn: dtree
@mx:mx_sum,kok:75.59%
0.01 s, 13:16:34 ,t0, 13:16:34
@msgn: svm
@mx:mx_sum,kok:71.53%
1.83 s, 13:16:36 ,t0, 13:16:34
@msgn: mlp
@mx:mx_sum,kok:47.95%
```

```
0.02 s, 13:10:03 ,t0, 13:10:03
@msgn: mlpreg
@mx:mx_sum,kok:99.96%
0.01s, 13:10:03 ,t0, 13:10:03
5%误差精度，yk0=5
```

```
0.01 s, 13:16:36 ,t0, 13:16:36
@msgn: mlpreg
@mx:mx_sum,kok:62.37%
0.01 s, 13:16:36 ,t0, 13:16:36
1%误差精度，yk0=1
```

从以上结果对比信息可以看出：

- 除了 SVM 向量机算法运行速度是 1.8 秒以外，其他的机器学习算法，速度都在零点零几秒左右，而且运行速度与 yk0 误差精度没有关系。
- 当 yk0=5，误差精度为 5%时，除 SVM 向量机算法准确度是 96%以外，其他算法的准确度都在 99%左右。
- 当 yk0=1，误差精度为 1%时，准确度度最高的是 Forest 随机森林算法 78.64%，最低的是 mlp 神经网络算法，只有 47.95%。
- 有趣的是，误差精度为 1%时，原本效果最差的 SVM 向量机算法准确度为 71.5%，名列前茅。

案例 8-10：机器学习组合算法

本章前面介绍了不少案例，其中大部分是为了给机器学习组合算法做铺垫。

机器学习组合算法，看起来简单，其实有以下几个难点。

- 如何集成多种机器学习算法模型，并进行测试。
- 如何评估多个算法模型的预测结果，并得出最佳结果数据。
- 采用什么样的机器学习算法组合最好。

针对以上难点，实盘往往会采用权重的模式，这样大大增加了算法和程序的复杂程度，对于实盘效果的提升也非常有限，所以我们采用统一权重，所有参数的权重都为 1。

实盘测试发现，并非组合的算法模型越多，最终的结果准确度越高，有时，模型种类多的算法组合的准确度，会低于种类少的算法组合。

这一点，在本章的案例中也会介绍，这也从机器学习组合算法的角度，证明了笔者小数据算法理论的正确性。

机器学习组合算法，也可以称为机器学习集成算法。

案例 8-10 文件名是 zai410_xsum.py，介绍如何使用机器学习集成算法，以及各种机器学习组合算法的性能测试，下面我们分组介绍相关的程序代码。

第 1 组代码，设置相关的参数，并读取测试所需的数据集：

```
    #1
    fsr0='dat/ccpp_'
    print('#1',fsr0)
    mlst1=['line','log','bayes','knn','forest','dtree','svm',
'mlp','mlpreg']
    mlst2=['forest','dtree','svm']
    mlst3=['line','log','bayes']
    mlst4=['line','forest','dtree']

    x_test=zai.ai_f_datRd010(fsr0+'xtest.csv')
    y_test=zai.ai_f_datRd010(fsr0+'ytest.csv',1)
```

在以上代码中，需要注意的是 mlst 多组列表变量，代表了不同的机器学习算法组合。

第 2 组代码，从文件中批量读取机器学习算法模型，并保存到全局变量 zai.xmodel：

```
    #2
    zai.xmodel={}
    print('\n#2,ai_f_mxRdlst')
    zai.ai_f_mxRdlst(fsr0,mlst1)
```

批量读取的输入参数，mlst 必须覆盖程序中所需的各种机器学习算法标识符，不过没有现成的列表变量，需要采用集合运算生成一个新的列表数据合集。

第 3 组代码，采用 mlst1 列表参数，调用 zai.mx_mul 机器学习组合算法函数：

```
    #3
    print('\n#3,mlst1',mlst1)
    zai.mx_mul(mlst1,  x_test,   y_test,yk0=1,fgInt=False,fgDebug=
False)
```

由于第 3 组至第 6 组代码基本相同，只是调用参数不同，所以我们合并相关的输出信息，一起解读。

第 3 组至第 6 组代码，对应的输出数据分别是：

```
#3,mlst1    ['line',    'log',
'bayes',    'knn',    'forest',
'dtree', 'svm', 'mlp', 'mlpreg']

y_pred,预测
1 y_pred01,kok:68.52% line 0.0 s
2 y_pred02,kok:53.64% log 0.0 s
3 y_pred03,kok:54.72% bayes 0.0
s
4 y_pred04,kok:66.14% knn 0.03 s
5  y_pred05,kok:78.64%  forest
0.02 s
6 y_pred06,kok:75.59% dtree 0.0
s
7 y_pred07,kok:71.53% svm 1.92 s
8 y_pred08,kok:47.95% mlp 0.01 s
9 y_pred09,kok:62.37% mlpreg 0.0
s
```

```
#4,mlst2  ['forest',  'dtree',
'svm']

y_pred,预测
1  y_pred01,kok:78.64%  forest
0.02 s
2 y_pred02,kok:75.59% dtree 0.0
s
3 y_pred03,kok:71.53% svm 1.9 s
@mx:mx_sum,kok:78.60%
```

第 4 组输出信息

```
@mx:mx_sum,kok:74.37%
第 3 组输出信息
#5,mlst3 ['line', 'log', 'bayes']

y_pred,预测
1 y_pred01,kok:68.52% line 0.0 s
2 y_pred02,kok:53.64% log 0.0 s
3 y_pred03,kok:54.72% bayes 0.0 s
@mx:mx_sum,kok:64.34%

第 5 组输出信息
```

```
#6,mlst3 ['line', 'forest', 'dtree']

y_pred,预测
1 y_pred01,kok:68.52% line 0.0 s
2  y_pred02,kok:78.64%  forest
0.02 s
3 y_pred03,kok:75.59% dtree 0.0 s
@mx:mx_sum,kok:81.98%
第 6 组输出信息
```

在以上输出信息中，mx_sum 代表的是机器学习组合算法的综合结果。

由以上结果可以看出：

- 准确度最高的是第 6 组，81.98%；准确度最低的是第 5 组，64.34%。
- 除了第 6 组以外，其他的组合算法最终结果都低于组合中的最高单个算法数值。

以上案例结果，也从机器学习组合算法的角度，证明了笔者小数据算法的合理性。

机器学习组合算法函数

zai.mx_mul 机器学习组合算法函数，位于在 ztop_ai 极宽机器学习模块中，函数代码如下：

```
    def  mx_mul(mlst,  x_test,   y_test,yk0=5,fgInt=False,fgDebug=
False):
        #1
        print('\ny_pred,预测')
        df9,xc,mxn9=x_test.copy(),0,len(mlst)
        df9['y_test']=y_test
        #2
        for msgn in mlst:
```

```
            xc+=1;tim0=arrow.now()
            mx=xmodel[msgn]
            y_pred = mx.predict(x_test.values)
            #3
            if xc==1:df9['y_sum']=y_pred
            else:df9['y_sum']=df9['y_sum']+y_pred
            #4
            tn=zt.timNSec('',tim0)
            df9['y_pred']=y_pred
            #5
            dacc=ai_acc_xed(df9,1,fgDebug)
            xss='y_pred{0:02},kok:{1:.2f}%'.format(xc,dacc);print
(xc,xss,msgn,tn,'s')
            ysgn='y_pred'+str(xc);df9[ysgn]=y_pred
        #6
        df9['y_pred']=df9['y_sum']/mxn9
        dacc=ai_acc_xed(df9,yk0,fgDebug)

        #7
        if fgDebug:
            df9.to_csv('tmp/df9_pred.csv');

        #8
        print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
        return dacc,df9
```

zai.mx_mul 机器学习组合算法函数的代码并不复杂，重点是第 2 组代码：

```
    for msgn in mlst:
```

通过迭代循环，计算各个机器学习算法的结果与准确度，并计算累加的总的准确度，直到第 6 组代码，对总的准确度进行平均计算。

其他的代码前面已经讲述过，此处不再赘述。

8.4 案例源码

本章包括以下案例程序。

- 案例 8-1：CCPP 数据集，文件名是 zai401_ccpp01.py。
- 案例 8-2：CCPP 数据切割，文件名是 zai402_ccpp02.py。
- 案例 8-3：读取 CCPP 数据集，文件名是 zai403_ccpp03.py。
- 案例 8-4：机器学习统一接口，文件名是 zai404_mx_01.py。
- 案例 8-5：批量调用，文件名是 zai405_mx_02.py。
- 案例 8-6：一体化调用，文件名是 zai406_mx_03.py。
- 案例 8-7：储存算法模型，文件名是 zai407_mx_04.py。
- 案例 8-8：批量储存算法模型，文件名是 zai408_mx_05.py。
- 案例 8-9：批量加载算法模型，文件名是 zai409_mx_06.py。
- 案例 8-10：机器学习组合算法，文件名是 zai410_xsum.py。

案例 8-1：CCPP数据集

案例 8-1：CCPP 数据集，文件名是 zai401_ccpp01.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com

'''

import pandas as pd
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
```

```
import ztools_data as zdat
import zpd_talib as zta

#
#-----------------------

#1
fss='dat/ccpp.csv'
df=pd.read_csv(fss,index_col=False)
print('\n#1 df')
print(df.tail())
print(df.describe())




#-----------------------
print('\nok!')
```

案例 8-2：CCPP数据切割

案例 8-2：CCPP 数据切割，文件名是 zai402_ccpp02.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com

'''



import pandas as pd
import sklearn
```

```
    from sklearn import datasets, linear_model
    from sklearn.cross_validation import train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn import metrics
    from sklearn.model_selection import cross_val_predict

    #
    import zsys
    import ztools as zt
    import ztools_str as zstr
    import ztools_web as zweb
    import ztools_data as zdat
    import zpd_talib as zta

    #
    #-----------------------
    def ai_data_cut(df,xlst,ysgn,ftg0,fgPr=False):
        x,y= df[xlst],df[ysgn]
        x_train, x_test, y_train, y_test = train_test_split(x, y,
random_state=1)
        #
        fss=ftg0+'xtrain.csv';x_train.to_csv(fss,index=False);
print(fss)
        fss=ftg0+'xtest.csv';x_test.to_csv(fss,index=False);
print(fss)
        fss=ftg0+'ytrain.csv';y_train.to_csv(fss,index=False,
header=True);print(fss)
        fss=ftg0+'ytest.csv';y_test.to_csv(fss,index=False,
header=True);print(fss)
        #
        if fgPr:
            print('\nx_train');print(x_train.tail())
            print('\nx_test');print(x_test.tail())
            print('\ny_train');print(y_train.tail())
```

```
        print('\ny_test');print(y_test.tail())




    #-----------------------


    #1
    fss='dat/ccpp.csv'
    df=pd.read_csv(fss,index_col=False)

    #2
    print('\n2# df')
    print(df.tail())

    #3
    xlst,ysgn=['AT','V','AP','RH'],'PE'
    ftg0='tmp/ccpp_'
    ai_data_cut(df,xlst,ysgn,ftg0,True)




    #-----------------------
    print('\nok!')
```

案例 8-3：读取CCPP数据集

案例 8-3：读取 CCPP 数据集，文件名是 zai403_ccpp03.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com
```

```
'''

import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import zpd_talib as zta
#
#----------------------

#1
fsr0='dat/ccpp_'
print('#2,',fsr0)

#2
k0=1
print('\n#2,k0',k0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0,k0,True)

#3
k0=10
print('\n#3,k0',k0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0,k0,True)
```

```
#----------------------
print('\nok!')
```

案例 8-4：机器学习统一接口

案例 8-4：机器学习统一接口，文件名是 zai404_mx_01.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''


import arrow
import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import zpd_talib as zta
```

```
    #
    #----------------------



    def mx_fun010(funSgn,x_train, x_test, y_train, y_test,yk0=5,
fgInt=False,fgDebug=False):
        #1
        df9=x_test.copy()
        mx_fun=zai.mxfunSgn[funSgn]
        mx =mx_fun(x_train.values,y_train.values)
        #2
        y_pred = mx.predict(x_test.values)
        df9['y_test'],df9['y_pred']=y_test,y_pred
        #3
        if fgInt:
            df9['y_predsr']=df9['y_pred']
            df9['y_pred']=round(df9['y_predsr']).astype(int)

        #4
        dacc=zai.ai_acc_xed(df9,yk0,fgDebug)
        #5
        if fgDebug:
            #print(df9.head())
            print('@fun name:',mx_fun.__name__)
            df9.to_csv('tmp/df9_pred.csv');
        #
        #6
        print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
        return dacc,df9

    #----------------------

    #1
    fsr0='dat/ccpp_'
```

```
    print('#1',fsr0)
    x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)

    #2
    print('\n#2,mx_line')
    funSgn='line'
    tim0=arrow.now()
    dacc,df9=mx_fun010(funSgn,x_train, x_test, y_train, y_test,5,
False)
    tn=zt.timNSec('',tim0,True)

    #3
    print('\n#3,mx_log')
    funSgn='log'
    tim0=arrow.now()
    dacc,df9=mx_fun010(funSgn,x_train, x_test, y_train, y_test,5,
False,True)
    tn=zt.timNSec('',tim0,True)



    #----------------------
    print('\nok!')
```

案例 8-5：批量调用

案例 8-5：批量调用，文件名是 zai405_mx_02.py，源码如下。

```
    #coding=utf-8
    '''
    Created on 2016.12.25
    TopQuant-极宽量化系统·培训课件-配套教学 python 程序
    @ www.TopQuant.vip      www.ziwang.com

```

```
    '''

    import arrow
    import pandas as pd
    import sklearn
    from sklearn import datasets, linear_model
    from sklearn.cross_validation import train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn import metrics
    from sklearn.model_selection import cross_val_predict

    #
    import zsys
    import ztools as zt
    import ztools_str as zstr
    import ztools_web as zweb
    import ztools_data as zdat
    import zpd_talib as zta

    #
    #-----------------------



    #1
    mxlst1=['line','knn','bayes']
    mxlst2=['line','log','bayes','knn','forest','dtree','svm','ml
p','mlpreg']
    #'gbdt','svmcr'

    fsr0='dat/ccpp_'
    print('#1',fsr0)
    x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)

    #2
```

```
print('\n#2,mxlst1')
zai.mx_funlst(mxlst1,x_train, x_test, y_train, y_test)

#3
print('\n#3,mxlst2')
zai.mx_funlst(mxlst2,x_train, x_test, y_train, y_test)



#-----------------------
print('\nok!')
```

案例 8-6：一体化调用

案例 8-6：一体化调用，文件名是 zai406_mx_03.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''


import arrow
import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
```

```
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import zpd_talib as zta
import ztop_ai as zai

#
#-----------------------




#1
xlst,ysgn=['AT','V','AP','RH'],'PE'
df=pd.read_csv('dat/ccpp.csv',index_col=False)

#2
print('\n#2,mx_line')
funsgn='line'
tim0=arrow.now()
zai.mx_fun_call(df,xlst,ysgn,funsgn)
tn=zt.timNSec('',tim0,True)




#-----------------------
print('\nok!')
```

案例 8-7：储存算法模型

案例 8-7：储存算法模型，文件名是 zai407_mx_04.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''



import arrow
import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.externals import joblib
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import zpd_talib as zta
import ztop_ai as zai



#------------------------

#1
fsr0='dat/ccpp_'
print('#1',fsr0)
```

```
xlst,ysgn=['AT','V','AP','RH'],'PE'
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
funSgn,ftg='svm','tmp/ccpp_svm.pkl'
#2
print('\n#2,mx_svm.wr')
tim0=arrow.now()
zai.ai_f_mxWr(ftg,funSgn,x_train, y_train)
tn=zt.timNSec('',tim0,True)

#3
tim0=arrow.now()
print('\n#3,mx_svm.rd')
mx = joblib.load(ftg)
tn=zt.timNSec('',tim0,True)

#4
print('\n#4,mx_svm')
tim0=arrow.now()
zai.mx_fun8mx(mx,x_test,y_test)
tn=zt.timNSec('',tim0,True)

#-----------------------
print('\nok!')
```

案例 8-8：批量储存算法模型

案例 8-8：批量储存算法模型，文件名是 zai408_mx_05.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip      www.ziwang.com
```

```
'''


import arrow
import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import zpd_talib as zta
import ztop_ai as zai

#
#-----------------------



#1
fsr0='dat/ccpp_'
print('#1',fsr0)
mxlst=['line','log','bayes','knn']
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
ftg0='tmp/ccpp_'

#2
```

```
print('\n#2,mx_svm.wr')
zai.ai_f_mxWrlst(ftg0,mxlst,x_train, y_train)

#-----------------------
print('\nok!')
```

案例 8-9：批量加载算法模型

案例 8-9：批量加载算法模型，文件名是 zai409_mx_06.py，源码如下。

```
#coding=utf-8
'''
Created on 2016.12.25
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''



import arrow
import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
```

```
    import ztools_data as zdat
    import zpd_talib as zta
    import ztop_ai as zai

    #----------------------


    #1
    fsr0='dat/ccpp_'
    print('#1',fsr0)
    mxlst=['line','log','bayes','knn','forest','dtree','svm','mlp
','mlpreg']
    x_test=zai.ai_f_datRd010(fsr0+'xtest.csv')
    y_test=zai.ai_f_datRd010(fsr0+'ytest.csv',1)

    #2
    zai.xmodel={}
    print('\n#2,ai_f_mxRdlst')
    zai.ai_f_mxRdlst(fsr0,mxlst)

    #3
    print('\n#3,mx_funlst8mx')
    zai.mx_funlst8mx(mxlst, x_test,  y_test,yk0=5,fgInt=False)

    #----------------------
    print('\nok!')
```

案例 8-10：机器学习组合算法

案例 8-10：机器学习组合算法，文件名是 zai410_xsum.py，源码如下。

```
    #coding=utf-8
    '''
    Created on 2016.12.25
```

```
TopQuant-极宽量化系统·培训课件-配套教学 python 程序
@ www.TopQuant.vip     www.ziwang.com

'''


import arrow
import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

#
import zsys
import ztools as zt
import ztools_str as zstr
import ztools_web as zweb
import ztools_data as zdat
import zpd_talib as zta
import ztop_ai as zai

#
#-----------------------


#-----------------------

#1
fsr0='dat/ccpp_'
print('#1',fsr0)
mlst1=['line','log','bayes','knn','forest','dtree','svm','mlp
','mlpreg']
```

```
    mlst2=['forest','dtree','svm']
    mlst3=['line','log','bayes']
    mlst4=['line','forest','dtree']

    x_test=zai.ai_f_datRd010(fsr0+'xtest.csv')
    y_test=zai.ai_f_datRd010(fsr0+'ytest.csv',1)

    #2
    zai.xmodel={}
    print('\n#2,ai_f_mxRdlst')
    zai.ai_f_mxRdlst(fsr0,mlst1)

    #3
    print('\n#3,mlst1',mlst1)
    zai.mx_mul(mlst1,  x_test,   y_test,yk0=1,fgInt=False,fgDebug=
False)

    #4
    print('\n#4,mlst2',mlst2)
    zai.mx_mul(mlst2,  x_test,   y_test,yk0=1,fgInt=False,fgDebug=
False)

    #5
    print('\n#5,mlst3',mlst3)
    zai.mx_mul(mlst3,  x_test,   y_test,yk0=1,fgInt=False,fgDebug=
False)

    #6
    print('\n#6,mlst3',mlst4)
    zai.mx_mul(mlst4,  x_test,   y_test,yk0=1,fgInt=False,fgDebug=
False)

    #-----------------------
    print('\nok!')
```



附录 A

Sklearn 常用模块和函数

本附录主要用于查询 Sklearn 人工智能、机器学习的相关函数。

1．模块库清单

Sklearn 人工智能、机器学习模块，常用的子模块库如下。

- sklearn.base：基础模块库。
- sklearn.utils：常用工具模块库。
- sklearn.cluster：聚类模块库。
- sklearn.cluster.bicluster：双聚类模块库。
- sklearn.covariance：协方差评估模块库。
- sklearn.model_selection：模型选择模块库。该模块库包括：
    - Splitter Classes：分割器类。
    - Splitter Functions：分割器函数。
    - Hyper-parameter：超参数优化。
    - optimizers Model validation：优化模型验证。
- sklearn.datasets：数据集模块库。
- sklearn.decomposition：矩阵分解模块库。
- sklearn.dummy：伪变量模块库。

- sklearn.ensemble：集成算法模块库。
- sklearn.exceptions；异常和警告模块库。
- sklearn.feature_extraction：特征提取模块库。
- sklearn.feature_selection；特征选择模块库。
- sklearn.gaussian_process：高斯过程模块库。
- sklearn.isotonic：保序回归模块库。
- sklearn.kernel_approximation；核近似模块库。
- sklearn.kernel_ridge；核岭回归模块库。
- sklearn.discriminant_analysis：判别分析模块库。
- sklearn.linear_model：线性模型模块库。
- sklearn.manifold：流形学习模块库。
- sklearn.metrics：结果测度模块库。该模块库包括：
  - Model Selection Interface：模型选择界面。
  - Classification metrics：分类度量。
  - Regression metrics：回归度量。
  - Multilabel ranking metrics：多类别排名度量。
  - Clustering metrics：聚类度量。
  - Biclustering metrics：双聚类度量。
  - Pairwise metrics：配对度量。
- sklearn.mixture：高斯混合模型模块库。
- sklearn.multiclass：多分类模块库。
- sklearn.multioutput：多输出回归和分类模块库。
- sklearn.naive_bayes：朴素贝叶斯算法模块库。
- sklearn.neighbors：近邻算法模块库。
- sklearn.neural_network：神经网络算法模块库。
- sklearn.calibration：校准概率模块库。
- sklearn.cross_decomposition：交叉分解算法模块库。
- sklearn.pipeline：管道数据记录模块库。
- sklearn.preprocessing：预处理和归一化模块库。

- sklearn.random_projection：随机投影算法模块库。
- sklearn.semi_supervised；半监督学习算法模块库。
- sklearn.svm：svm 向量机算法模块库。
- sklearn.tree：决策树算法模块库。

2. Base基础模块

Base 基础模块库 sklearn.base，包括 Base 基类定义和工具函数，常用类定义如下。

- BaseEstimator：测试评估基类。
- ClassifierMixin：混合分类器。
- ClusterMixin：混合聚类等基类。
- RegressorMixin：混合回归评估基类。
- TransformerMixin：混合转换基类。

sklearn.base 常用函数是 base.clone，即复制一个新的基类变量。

3. Utils常用工具模块

sklearn.utils：常用工具模块库，包括常用的工具函数和各种实用程序。

- check_random_state：生成随机 Randomstate 实例。
- estimator_checks.check_estimator：检查评估参数。
- resample：数组或稀疏矩阵，重采样。
- shuffle：数组或稀疏矩阵混洗整理。

4. Exceptions异常和警告

sklearn.exceptions：异常和警告模块库，包括 Sklearn 模块库所有的警告、出错信息和处理函数。

- NotFittedError：如果分类器在拟合前就使用，则抛出异常。
- ChangedBehaviorWarning：警告类通知，用户有任何行为上的变更。
- ConvergenceWarning：自定义警告，来捕捉收敛问题

- DataConversionWarning：警告通知，用于隐式数据转换代码。
- DataDimensionalityWarning：自定义警报，通知用户数据维度的潜在问题。
- EfficiencyWarning：警告通知用户，计算效率偏低。
- FitFailedWarning：警告类，用于拟合评估时存在的误差。
- NonBLASDotWarning：当没使用 BlAS 就计算点运算时，出现警告。
- Undefinedmetricwarning：无效度量时出现警告。

5. Cluster聚类模块

sklearn.cluster：无监督聚类算法模块库，主要的类定义如下。

- AffinityPropagation：仿射传播聚类算法。
- AgglomerativeClustering：凝聚聚类算法。
- BIRCH：聚类算法。
- DBSCAN：矢量聚类算法（根据矩阵距离）。
- FeatureAgglomeration：团簇特征聚类算法。
- KMeans：K-均值聚类算法。
- MiniBatchKMeans：迷你批次 K-均值聚类算法。
- MeanShift：均值偏移聚类算法。
- SpectralClustering：投影归一化（拉普拉斯算子）聚类算法。

主要的函数如下。

- estimate_bandwidth：评估均值偏移算法使用的带宽。
- k_means：　K-means 均值聚类算法。
- ward_tree：沃德基于聚类的特征矩阵。
- affinity_propagation：仿射传播聚类算法。
- dbscan：DBSCAN 密度聚类算法。
- mean_shift：均值偏移聚类算法。
- spectral_clustering：投影归一化（拉普拉斯算子）　聚类算法。

6. cluster.bicluster双聚类

sklearn.cluster.bicluster：双聚类模块库，主要包括以下算法。

- SpectralBiclustering：Kluger 双谱聚类算法。
- SpectralCoclustering：Dhillon 同谱聚类算法。

7. Covariance协方差评估

sklearn.covariance：协方差评估模块，包括一组稳健特征协方差评估算法。精度矩阵定义为逆的协方差，协方差评估与高斯图形模型理论密切相关。

包括以下函数。

- EmpiricalCovariance：最大似然评估量协方差。
- EllipticEnvelope：用于检测对象的离群值的高斯分布数据集。
- GraphLasso：稀疏逆协方差评估算法，使用 L1 惩罚评估器。
- GraphLassoCV：稀疏逆协方差交叉验证，使用 L1 惩罚评估器。
- LedoitWolf：估计量。
- MinCovDet：最小协方差行列式（MCD），稳健评估量的协方差。
- OAS：Oracle 逼近收缩评估量。
- ShrunkCovariance：协方差收缩评估算法。
- empirical_covariance：计算最大似然评估量协方差。
- ledoit_wolf：协方差矩阵评估算法。
- shrunk_covariance：矩阵对角线协方差计算。
- OAS：中心评估协方差与 Oracle 逼近收缩算法。
- graph_lasso：L1 处罚协方差评估算法。

8. model_selection模型选择

sklearn.model_selection：模型选择模块库，包括以下几大部件。

- Splitter Classes：分割器类。

- Splitter Functions：分割器函数。
- Hyper-parameter：超参数优化。
- optimizers Model validation：模型验证优化。

（1）Splitter Classes 分割器类

- KFold：K-折交叉验证。
- GroupKFold：K-折迭代变量的非重叠组。
- StratifiedKFold：分层 K-折交叉验证算法。
- LeavePGroupsOut：留一组交叉验证算法。
- leavepgroupsoutN：留 P 组交叉验证程序。
- LeaveOneOut：留一法交叉验证程序。
- LeavePOut：留 P 法交叉验证算法。
- ShuffleSplit：随机置换交叉验证算法。
- GroupShuffleSplit：混洗组交叉迭代验证算法。
- StratifiedShuffleSplit：分层混洗分割交叉验证程序。
- PredefinedSplit：预定义分割交叉验证程序。
- TimeSeriesSplit：时间序列交叉验证程序。

（2）Splitter Functions 分割函数

- train_test_split：训练和测试数据集分割函数。
- check_cv：建立交叉验证输入检查器。

（3）Hyper-parameter 超参数优化

- GridSearchCV：网格搜索交叉验证。
- RandomizedSearchCV：随机搜索交叉验证。
- ParameterGrid：对每一个离散值进行参数网格搜索。
- ParameterSampler：指定分布按参数抽样生成器。
- fit_grid_point：按一组参数运行拟合。

（4）optimizers Model validation 模型验证优化

- cross_val_score：交叉验证评估得分。
- cross_val_predict：对每一个输入数据点生成交叉验证评估器。
- permutation_test_score：评估置换交叉验证测试得分的重要性。
- learning_curve：学习曲线。
- validation_curve：验证曲线。

9. datasets数据集模块

sklearn.datasets：数据集模块库模块，包括一些常用的测试数据，以及数据的加载函数，同时还提供一些人工数据生成器函数。

（1）Loader 数据加载

- clear_data_home：清除所有加载的数据。
- get_data_home：返回 scikit-learn 数据 dir 路径。
- fetch_20newsgroups：抓取 20 组新闻数据集的文件名和数据。
- fetch_20newsgroups_vectorized：抓取 20 组新闻数据集的 TF-IDF 向量化数据。
- load_boston：加载波士顿房屋价格数据集（回归）。
- load_breast_cancer：加载威斯康星州乳腺癌数据集（分类）。
- load_diabetes：加载糖尿病数据集（回归）。
- load_digits：加载数字 OCR 识别图像数据集（分类）。
- load_files：从类别子文件夹加载文本文件。
- load_iris：加载 Iris 爱丽丝经典数据集（分类）。
- fetch_lfw_pairs：加载 LFW 人脸两两数据集。
- fetch_lfw_people，LFW 人脸数据集。
- load_linnerud：加载 linnerud 多元回归数据集。
- mldata_filename：从 mldata.org 网站抓取数据集文件。
- fetch_mldata：从 mldata.org 网站抓取数据集。
- fetch_olivetti_faces：加载 Olivetti 面部数据集。

- fetch_california_housing：加载加州房屋数据集。
- fetch_covtype：加载并下载 covertype 数据集。
- fetch_kddcup99：加载 kddcup99 数据集。
- fetch_rcv1：加载并下载 RCV1 多标签数据集。
- load_mlcomp：因 mlcomp.org 数据网站将在 2017 年 3 月关闭，本加载函数在 0.19 和 0.21 版将被删除。
- load_sample_imageimage_name：采用 NumPy 数组加载单个图像样本。
- load_sample_images：加载 NumPy 数组格式的样本图像。
- fetch_species_distributions：加载物种分布数据集，来自 Phillips 等机构。
- load_svmlight_file：从文件加载 SVMLight/libsvm 文件格式的的数据，并转换为 CSR 稀疏矩阵格式。
- load_svmlight_files：从多个文件加载 SVMLight/libsvm 文件格式的数据，并转换为 CSR 稀疏矩阵格式。
- dump_svmlight_file：把数据集转为 SVMLight/libsvm 文件格式，并保存。

（2）样本数据生成器

- make_blobs：生成同性高斯斑点 blob 聚类数据。
- make_classification：产生随机 *N* 类分类数据。
- make_circles：生成二维包含指定的多个小圆的大圆圈数据。
- make_friedman1：生成的“弗里德曼#1”回归数据集。
- make_friedman2：生成的“弗里德曼#2”回归数据集。
- make_friedman3：生成的“弗里德曼#3”回归数据集。
- make_gaussian_quantiles：生成同性高斯样本分位数和标签。
- make_hastie_10_2：生成 Hastie 等二进制分类数据。
- make_low_rank_matrix：生成具有钟形奇异值的低秩矩阵。
- make_moons：生成两个交错半圆。
- make_multilabel_classification：生成随机多标记分类数据。
- make_regression：生成随机回归数据。

- make_s_curve：生成 S 曲线数据集。
- make_sparse_coded_signal：生成稀疏组合信号的字典数据。
- make_sparse_spd_matrix：生成 SPD 稀疏对称正定矩阵。
- make_sparse_uncorrelated：生成随机不相关稀疏数据集。
- make_spd_matrix：生成 SPD 随机对称正定矩阵。
- make_swiss_roll：生成瑞士卷数据集。
- make_biclusters：生成双聚类块对角结构矩阵。
- make_checkerboard：生成块结构的双聚类棋盘矩阵。

10. decomposition矩阵分解

sklearn.decomposition：矩阵分解模块库，包括矩阵分解模块算法，例如 PCA、ICA 或 NMF 等算法，该模块可作为降维使用。

- PCA：主成分分析。
- IncrementalPCA：增量主成份分析。
- KernelPCA ：核主成分分析（KPCA）。
- FactorAnalysis：因子分析（FA）。
- FastICA：一种快速独立分量分析算法。
- TruncatedSVD：使用截断奇异值分解降维（又称作 LSA）。
- NMF：非负矩阵分解。
- SparsePCA：稀疏主成分分析。
- MiniBatchSparsePCA：小批量稀疏主成分分析。
- SparseCoder：稀疏编码。
- DictionaryLearning：学习词典。
- MiniBatchDictionaryLearning：小批量学习词典。
- LatentDirichletAllocation：潜在狄利克雷分配与变分贝叶斯网络算法。
- Fastica：快速独立分量分析。
- dict_learning：解决学习字典矩阵分解问题。
- dict_learning_online：在线解决学习字典矩阵分解问题。
- sparse_encode：稀疏编码。

11. Dummy伪变量

sklearn.dummy：伪变量模块库。

- DummyClassifier：简单规则预测算法。
- DummyRegressor：简单规则回归预测算法。

12. Ensemble集成算法

sklearn.ensemble：集成算法模块，包括集成分类算法、回归算法和异常检测算法。

- AdaBoostClassifier：AdaBoost 迭代分类算法，其核心思想是针对同一个训练集训练不同的分类器（弱分类器），然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器）。
- AdaBoostRegressor：AdaBoost 回归分类算法。
- BaggingClassifier：Bagging 装袋算法，相当于多个专家投票表决，对于多次测试，每个样本返回的是多次预测结果较多的那个。
- BaggingRegressor：Bagging 装袋回归分类算法。
- ExtraTreesClassifier：完全随机树算法。
- ExtraTreesRegressor：完全随机树回归算法。
- GradientBoostingClassifier：GBT 梯度 Boosting 树增强算法。
- GradientBoostingRegressor：GBT 梯度 Boosting 树增强回归算法。
- IsolationForest：隔离森林算法。
- RandomForestClassifier：随机森林算法
- RandomTreesEmbedding：完全随机树算法。
- RandomForestRegressor：随机森林回归算法。
- VotingClassifier：投票算法，多种机器学习算法的集成式算法。

部分依赖 Tree 决策树的算法如下。

- partial_dependence.partial_dependence：局部变量依赖算法。
- partial_dependence.plot_partial_dependence：局部变量依赖算法绘图。

13. feature_selection特征选择

sklearn.feature_selection：特征选择模块，包括各种特征选择算法，如单变量特征选择算法、滤波器特征选择算法、递归特征消除算法等。

- GenericUnivariateSelect：通用单变量特征选择器的配置策略。
- SelectPercentile：按百分比最高分选择特征。
- SelectKBest：按最佳 k 值选择特征。
- SelectFpr：滤波器，在 FPR 测试中选择低于 alpha 的 p 值。
- SelectFdr：滤波器，为评估错误发现率选择 p 值。
- SelectFromModel：元数据转换选择特征算法，基于输入的权重数据。
- SelectFwe：过滤器，根据多重比较错误率选择 p 值。
- RFE：递归特征消除算法的特征排序。
- RFECV：递归特征消除算法的交叉验证特征排序。
- VarianceThreshold：特征选择算法，除去所有低方差特征。
- chi2：计算卡方统计每个非负特征和类。
- f_classif：计算样品的方差分析 F-值。
- f_regression：一元线性回归测试。
- mutual_info_classif：评估离散目标变量的互信息。
- mutual_info_regression：评估连续目标变量的互信息。

14. feature_extraction特征提取

sklearn.feature_extraction：特征提取模块，主要用于从原始数据提取特征数据，包括提取数值、文本、图像等数据的特征提取。

- DictVectorizer：变换的特征值映射为向量。
- FeatureHasher：把特征值采用 Hash 哈希函数进行散列变换。

（1）图像特征提取

sklearn.feature_extraction.image：从图像中提取特征。

- image.img_to_graph：像素-像素梯度连接图表。

- image.grid_to_graph：像素-像素连接网格图表。
- image.extract_patches_2d：把 2D 图像重塑为小块模块容器。
- image.reconstruct_from_patches_2d：从小块模块容器重建图像。
- image.PatchExtractor：从图像容器中抽取小块。

（2）文本特征提取

sklearn.feature_extraction.text 模块：从文本信息建立特征向量。

- text.CountVectorizer：将文本集合装换为 token 关键词统计向量矩阵。
- text.HashingVectorizer：采用哈希算法，将文本装换为向量矩阵。
- text.TfidfTransformer：把统计矩阵转换为归一化 tf 或 tf-idf 表征。
- text.TfidfVectorizer：把原文档转换为 tf-idf 特征矩阵。

15. gaussian_process高斯过程

sklearn.gaussian_process 模块实现了基于高斯过程回归和分类算法。

- GaussianProcessRegressor：高斯过程回归（GPR）算法。
- GaussianProcessClassifier：基于 Laplace 逼近的高斯过程分类（GPC）算法。

（1）Kernels 子模块功能：

- kernels.Kernel：核的基类。
- kernels.Sum：k1、k2 两个核求和运算。
- kernels.Product：k1、k2 两个核乘积运算。
- kernels.Exponentiation：乘方运算，根据给定的指数运算。
- kernels.ConstantKernel：核常量。
- kernels.WhiteKernel：白色核。
- kernels.RBF：核半径函数，径向基核函数 （又名平方指数内核）。
- kernels.Matern：母体核。
- kernels.RationalQuadratic：有理二次核。
- kernels.ExpSineSquared：正弦平方核。

- kernels.DotProduct：点积核。
- kernels.PairwiseKernel：sklearn.metrics.pairwise 内的核封装。
- kernels.CompoundKernel：由一组其他核构建的核。
- kernels.Hyperparameter：命名元组形式的超参数内核规范。

16. isotonic保序回归模块库。

sklearn.isotonic 模块：保序回归算法。

- IsotonicRegression：保序回归模型。
- isotonic_regression：保序回归模型求解算法。
- check_increasing(x, y)：Y 与 X 的单调相关确认算法。

17. kernel_approximation核近似模块

sklearn.kernel_approximation 模块收录了核近似算法，基于傅里叶变换特征映射内核近似算法。

- AdditiveChi2Sampler：添加 Chi2 卡方核近似特征映射。
- Nystroem：奈斯特龙 Nystroem 内核近似算法，使用训练数据的一个映射子集。
- RBFSampler：RBF 内核近似算法，基于蒙特卡洛傅里叶变换的 RBF 函数映射。
- SkewedChi2Sampler：近似特征映射的“斜卡方”内核，蒙特卡洛傅里叶变换算法。

18. kernel_ridge核岭回归模块库。

klearn.kernel_ridge 模块：实现核岭回归算法。

- KernelRidge：内核岭回归算法。

19. discriminant_analysis判别分析

sklearn.discriminant_analysis 模块：判别分析算法。

- LinearDiscriminantAnalysis：线性判别分析算法。
- QuadraticDiscriminantAnalysis：二次判别分析算法。

20. linear_model线性模型

sklearn.linear_model 模块库：通用线性模型算法，包括岭回归算法、套索回归算法、贝叶斯算法、弹性算法与最小二乘回归算法等，它还实现了相关的随机梯度下降算法。

- ARDRegression：ARD 贝叶斯算法。
- BayesianRidge：贝叶斯岭回归算法。
- ElasticNet：使用 L1 和 L2 正则的线性回归算法。
- ElasticNetCV：正则路径迭代拟合的交叉验证弹性网络。
- HuberRegressor：稳健离群值的线性回归算法。
- Lars：拉斯最小角回归算法。
- LarsCV：交叉验证最小角回归算法。
- Lasso：采用 L1 正则训练数据的线性算法（又称 Lasso 套索）。
- LassoCV：正则路径迭代拟合的交叉验证 Lasso 套索线性算法。
- LassoLars：套索与最小角度拟合回归算法。
- LassoLarsCV：交叉验证套索与最小角度拟合回归算法。
- LassoLarsIC：套索与与最小角度拟合回归算法，使用 AIC 或 BIC 选择模型。
- LinearRegression：普通最小二乘线性回归算法。
- LogisticRegression，逻辑回归算法（又称 logit.MaxEnt）。
- LogisticRegressionCV：交叉验证逻辑回归算法（又称 logit.MaxEnt）。
- MultiTaskLasso：多任务 Lasso 套索算法，使用 L1/L2 混合正则。
- MultiTaskElasticNet：多任务弹性网络算法，使用 L1/L2 混合正则。
- MultiTaskLassoCV：交叉验证 Lasso 套索算法。
- MultiTaskElasticNetCV：交叉验证多任务弹性网络算法。
- OrthogonalMatchingPursuit：正交匹配追踪（OMP）算法。
- OrthogonalMatchingPursuitCV：交叉验证正交匹配追踪（OMP）算法。

- PassiveAggressiveClassifier：被动的分类算法。
- PassiveAggressiveRegressor：被动侵略性回归算法。
- Perceptron：感知器算法。
- RandomizedLasso：随机 Lasso 套索算法。
- RandomizedLogisticRegression：随机逻辑回归算法。
- RANSACRegressor：RANSAC（RANdom SAmple Consensus）算法，统一随机采样算法。
- Ridge：岭回归算法。
- RidgeClassifier：岭回归分类算法。
- RidgeClassifierCV：交叉验证岭回归分类算法。
- RidgeCV：交叉验证岭回归算法。
- SGDClassifier：线性分类算法（支持向量机，逻辑回归等），使用随机梯度下降算法算法训练数据。
- SGDRegressor：正则化线性拟合算法，采用最小化经验损失与随机梯度下降算法。
- TheilSenRegressor：Theil-Sen 算法，采用稳健多变量回归模型。
- lars_path：最小角回归算法，使用套索路径或 LARS 拉斯算法。
- lasso_path：计算索套路径坐标下降数据。
- lasso_stability_path：基于随机套索稳定性评估的稳定路径算法。
- logistic_regression_path：计算逻辑回归模型，用于正则化参数列表。
- orthogonal_mp：正交匹配追踪（OMP）算法。
- orthogonal_mp_gram：gram 正交匹配追踪（OMP） 算法。

21. Manifold流形学习

sklearn.manifold 模块：流形学习算法、数据嵌入技术等。

- LocallyLinearEmbedding：局部线性嵌入算法。
- Isomap：映射嵌入算法。
- MDS：多维缩放算法。
- SpectralEmbedding：非线性频谱嵌入算法。

- TSNE：t 分布随机近邻嵌入算法。
- locally_linear_embedding：执行局部线性嵌入算法分析数据。
- spectral_embedding：基于图拉普拉斯第一特征向量的投影样本算法。
- SMACOF：多维度计算算法。

22. Mixture高斯混合模型

sklearn.mixture 模块：高斯建模算法。

- GaussianMixture：高斯混合算法。
- BayesianGaussianMixture：高斯贝叶斯混合算法。

23. Multiclass多分类

- OneVsRestClassifier：简称 OvR （1-VS-rest），一对余的多分类/多标签策略算法。
- OneVsOneClassifier：简称 OvO，一对一多分类策略算法。
- OutputCodeClassifier（纠错码）：输出多分类策略算法。

24. Multioutput多输出回归和分类

sklearn.multioutput 模块：实现多输出回归和分类算法。

- MultiOutputRegressor：多目标回归算法。
- MultiOutputClassifier：多目标分类算法。

25. naive_bayes朴素贝叶斯算法

sklearn.naive_bayes 模块，包括朴素贝叶斯 Bayes 算法，这些是在监督学习方法的基础上，运用贝叶斯定理和强（朴素）特征独立性假设而开发的算法。

- GaussianNB：高斯朴素贝叶斯算法。
- MultinomialNB：多项式朴素贝叶斯算法。
- BernoulliNB：多元伯努利（Bernoulli）朴素贝叶斯算法。

26. Neighbors近邻算法

sklearn.neighbors 模块：实现了 K-近邻算法。

- NearestNeighbors：无监督学习邻居搜索算法。
- KNeighborsClassifier：K-近邻算法。
- RadiusNeighborsClassifier：给定半径内的近邻投票算法。
- KNeighborsRegressor：K-近邻回归算法。
- RadiusNeighborsRegressor：固定半径内的近邻回归算法。
- NearestCentroid：最近质心算法。
- BallTree：BallTree 快速广义 N-点问题算法。
- KDTree：KDTree 快速广义 N-点问题算法。
- LSHForest：基于 LSH 森林模型的近邻算法。
- DistanceMetric：距离度量类。
- KernelDensity：核密度估计。
- LocalOutlierFactor：LOF 算法，局部异常因子的无监督异常检测算法。
- kneighbors_graph：计算（加权）为点 x 的 k 近邻图。
- radius_neighbors_graph：计算（加权）点半径为 X 的近邻图。

27. neural_network神经网络

sklearn.neural_network 模块：包括基于神经网络模型算法。

- BernoulliRBM：RBM 算法，伯努利受限玻尔兹曼机算法，也称有限伯努利机算法。
- MLPClassifier：多层感知算法。
- MLPRegressor：多层感知回归算法。

28. Calibration校准概率

sklearn.calibration 模块：包括预测概率的校准算法。

- CalibratedClassifierCV：保序回归或 sigmoid 概率校准算法。
- calibration_curve：计算真实预测概率的校正曲线。

29. cross_decomposition交叉分解算法

sklearn.cross_decomposition 模块：交叉分解算法。

- PLSRegression：PLS 最小二乘回归算法。
- PLSCanonical：PLS-C2A 算法。
- CCA：典型相关分析算法。
- PLSSVD：偏最小二乘奇异分解算法。

30. Pipeline管道数据记录

sklearn.pipeline 模块：用于记录 Sklearn 机器学习算法的各个流程。

- Pipeline：管道转换数据。
- FeatureUnion：串接多个转换器对象的结果。
- make_pipeline：按给定转换器构建管道。
- make_union：按给定转换器构建联合特征。

31. Preprocessing预处理和归一化

sklearn.preprocessing 模块：包括数据缩放、中心化、正则化、二元化及插补方法。

- Binarizer：按阀值对数据二元化（特征设为 0 或 1）。
- FunctionTransformer：从任意调度中构建转换器。
- Imputer：完成缺失数据插补。
- KernelCenterer：中心化核矩阵。
- LabelBinarizer：在一对剩余范式中二元化标签。
- LabelEncoder：用 0 到 n_class-1 对标签编码。
- MultiLabelBinarizer：对迭代器和多标签格式之间的转换器。
- MaxAbsScaler：对每一个特征按最大绝对值缩放。
- MinMaxScaler：对每一个特征按给定范围值缩放。
- Normalizer：分别对数据单位范数。

- OneHotEncoder：使用热码的整数编码分类特征，用 one-hot 又名 one-of-k 方案。
- PolynomialFeatures：生成多项式和交互特征。
- RobustScaler：使用统计缩放特征，对离群值 较稳健。
- StandardScaler：标准缩放特征，通过删除均值来缩放至单位方差。
- add_dummy_feature：通过添加伪特征增强数据集。
- binarize：布尔阀值化的数组或 scipy 稀疏矩阵。
- label_binarize：在一对多范式中二元化标签。
- maxabs_scale：不破坏稀疏性的情况下对每个特征缩放至[-1,1]。
- minmax_scale：对每个特征按给定范围值缩放。
- normalize：缩放输入向量至单位范数。
- robust_scale：沿任意轴标准化数据。
- scale：沿任意轴标准化数据。

32. random_projection随机投影算法

sklearn.random_projection 模块：包括数据缩放、归一化、二值化和估算方法。

- GaussianRandomProjection：随机高斯投影降维算法。
- SparseRandomProjection：稀疏随机投影降维算法。
- johnson_lindenstrauss_min_dim：找到“安全”数目的分量随机项目。

33. semi_supervised半监督学习算法

sklearn.semi_supervised 模块实现了半监督学习算法，这些算法利用少量的标记数据和大量的未标记的数据分类任务，包括标签传播算法。

- LabelPropagation：标签传播算法。
- LabelSpreading：半监督学习的标签传播算法。

34. SVM向量机算法

sklearn.svm 模块包括支持向量机算法。

- SVC：C-支持向量分类算法。
- LinearSVC：线性支持向量分类算法。
- NuSVC：Nu 支持向量分类算法。
- SVR：Epsilon 支持向量回归算法。
- LinearSVR：线性向量回归算法。
- NuSVR：Nu 支持向量回归算法。
- OneClassSVM：无监督离群点检测。
- l1_min_c：返回最低边界的 C(l1_min_C,infinity)，保证模型为非空。

35. Tree决策树算法

sklearn.tree 模块：包括决策树分类和回归算法模型。

- DecisionTreeClassifier：决策树分类算法。
- DecisionTreeRegressor：决策树回归算法。
- ExtraTreeClassifier：完全随机树分类算法。
- ExtraTreeRegressor：完全随机树回归算法。
- export_graphviz：采用 DOT 格式输出决策树图。

36. sklearn.metrics结果测度

sklearn.metrics 度量模块，包括模型、算法评估等功能，有评分功能、性能指标和对指标和距离计算。

包括以下运行部件。

- Model Selection Interface：模式选择界面。
- Classification metrics：分类度量。
- Regression metrics：回归度量。
- Multilabel ranking metrics：多标签排序度量。

- Clustering metrics：聚类度量。
- Biclustering metrics：双聚类度量。
- Pairwise metrics：成对度量。

（1）Model Selection Interface 模式选择界面

- make_scorer：按性能测试或损失函数评分。
- get_scorer：获取性能测试评分。

（2）Classification metrics 分类度量

- accuracy_score：准确性得分。
- AUC：采用梯形法计算曲线下面积 。
- average_precision_score：从预测评分计算平均精确度（AP）。
- brier_score_loss：计算 Brier 布莱尔分数。Brier 分数是评价频率预测准确度的一个指标，适用的前提是只关心某个事件是否发生（0~1 频度），而不关心事件发生的强度影响。
- classification_report：建立一个主要分类指标的文本报告。
- cohen_kappa_score：科恩 Kappa 评分，统计测量注释协议。
- confusion_matrix：计算混淆矩阵来评估分类准确性。
- f1_score：计算 F1 得分，又称为平衡 F 或 F-分数度量。
- fbeta_score：计算 F-beta 得分。
- hamming_loss：计算平均汉明损失。
- hinge_loss：平均连接损失（未正则化）。
- jaccard_similarity_score：Jaccard 相似性系数评分。
- log_loss：log 损失，逻辑损失，又称作交叉熵损失。
- matthews_corrcoef：计算二分类的马修斯相关系数（MCC）。
- precision_recall_curve：对不同概率阈值精度计算-召回率。
- precision_recall_fscore_support：计算准确率、查全率、F-measure 等参数。
- precision_score：计算精度。
- recall_score：计算召回率。

- roc_auc_score：计算曲线下面积（AUC）的预测评分。
- roc_curve：计算 ROC 受试者操作特征评分。
- zero_one_loss：零分类损失。

（3）Regression metrics 回归度量

- explained_variance_score：解释方差回归得分函数。
- mean_absolute_error：平均绝对误差回归损失。
- mean_squared_error：回归均值平方误差损失。
- mean_squared_log_error：对数回归均值平方误差损失。
- median_absolute_error：中位数绝对误差回归损失。
- r2_score：R 方确定系数的回归得分函数。

（4）Multilabel ranking metrics 多标签排序度量

- coverage_error：覆盖误差测量。
- label_ranking_average_precision_score：计算平均精度等级。
- label_ranking_loss：计算排序度量损失。

（5）Clustering metrics 聚类度量

sklearn.metrics 模块：对于分析结果的度量评估，采用以下两种评估形式。

- 监督：采用每个样本的真实类别。
- 无监督：没有采用每个样本的真实类别或测量模型本身的“质量”数据。

相关函数如下。

- adjusted_mutual_info_score：两个簇的调整型互信息。
- adjusted_rand_score：概率型随机索引调整。
- calinski_harabaz_score：计算 Calinski 和 harabaz 评分。
- completeness_score：给定真实值对簇标签的完整性度量。
- fowlkes_mallows_score：测量两组点集簇的相似性。
- homogeneity_completeness_v_measure：一次性计算同质性和完整性度量，v-measure 评分。

- homogeneity_score：给定真实值对簇标签的同质性度量。
- mutual_info_score：两个簇间的互信息。
- normalized_mutual_info_score：两个簇间的归一化互信息。
- silhouette_score：计算所有样品的平均轮廓系数。
- silhouette_samples：计算各样本的轮廓系数。
- v_measure_score：给定真实值的簇标签 V-测量。

（6）Biclustering metrics 双聚类度量

- metrics.consensus_score：双聚类的相似性得分。

（7）Pairwise metrics 配对度量

配对度量，也称为两两度量。

- pairwise.additive_chi2_kernel：计算观测 X 和 Y 间的附加卡方核数据。
- pairwise.chi2_kernel：计算 X 和 Y 的卡方核指数。
- pairwise.distance_metrics：成对有效距离度量。
- pairwise.euclidean_distances：计算两两向量间的距离矩阵。
- pairwise.kernel_metrics：两两核的有效性度量。
- pairwise.linear_kernel：计算的 X 和 Y 之间的线性核。
- pairwise.manhattan_distances：计算各向量间的 L1 距离。
- pairwise.pairwise_distances：计算向量数组 X 和可选数组 Y 的距离矩阵。
- pairwise.pairwise_kernels：计算数组 X 和可选数组 Y 之间的核。
- pairwise.polynomial_kernel：计算 X 和 Y 之间的多项式核。
- pairwise.rbf_kernel：计算 X 和 Y 之间的高斯（RBF）核。
- pairwise.sigmoid_kernel：计算 X 和 Y 之间的 Sigmoid 核。
- pairwise.cosine_similarity：计算 X 和 Y 之间的余弦相似度。
- pairwise.cosine_distances：计算 X 和 Y 之间的余弦距离。
- pairwise.laplacian_kernel：计算 X 和 Y 之间的拉普拉斯核。
- pairwise_distances：计算向量数组 X 和可选 Y 之间的距离矩阵。

- pairwise_distances_argmin：计算一个点集和一组点集的最小距离。
- pairwise_distances_argmin_min：计算一个点集和一组点集的最小距离。pairwise.paired_euclidean_distances：计算 X 和 Y 之间的两两欧氏距离。
- pairwise.paired_manhattan_distances：计算 X 和 Y 之间的两两 L1 距离。
- pairwise.paired_cosine_distances：计算 X 和 Y 之间的两两余弦距离。
- pairwise.paired_distances：计算 X 和 Y 之间的两两距离。

附录 B

B

极宽量化系统模块图

为方便读者进一步学习，下面对于 TopQuant 极宽金融量化系统、TFB 极宽足彩量化分析系统和极宽底层模块，做一个简单的介绍。

1. TFB足彩量化系统

Top Football 极宽足彩量化分析系统，又称 Top Quant for football，简称 TFB。

TFB 极宽足彩量化分析系统，模块构成如图 B-1 所示。



图 B-1　TFB 模块构成图

由图 B-1 可以看出，TFB 主要有以下三大模块构成。

- Top-Base：极宽基础模块库。
- Top-Football：极宽足彩专业模块库。
- tfbDat：极宽足彩数据包。

2. Top-Base极宽基础模块库

Top-Base 极宽基础模块库是极宽各种系统的基础模块库，各个模块文件均采用 z 字母作为文件首字母，一方面是为了和各个应用模块区别，另外一方面是为了表示极宽系统源自最初的 zw 量化系统。

如图 B-2 所示是 Top-Base 极宽基础模块库的主要模块组成图。

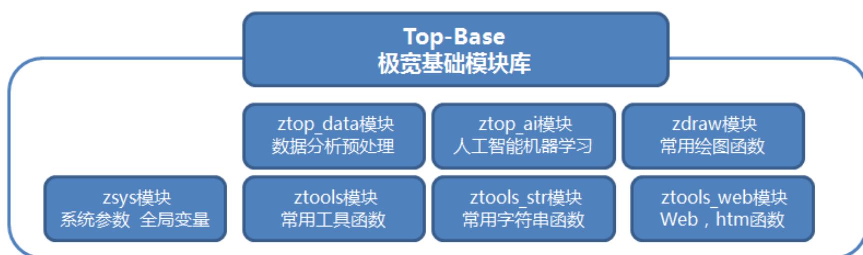


图 B-2　Top-Base 极宽基础模块库组成图

Top-Base 极宽基础模块库包括以下模块。

- zsys：全局系统模块，定义全局参数、变量、常数等。
- ztools：常用工具函数库，缩写为 zt。
- ztools_str：常用字符串函数库，缩写为 zstr。
- ztools_web：常用 Web 网页抓取，htm 分析函数库，缩写为 zweb。
- zdraw：常用绘图函数库。
- ztop_data：极宽数据分析、预处理函数库，缩写为 zdat。
- ztop_ai：极宽通用机器学习模块库，缩写为 zai。

需要说明的是，随着未来极宽各个应用软件和系统的升级和发展，各个应用模块的构成可能会有所变动，请大家注意。

此外，因为字符串处理和 Web 网页数据抓取分析，使用频率较多，所以从原来的 ztools 常用工具函数库分离出来。

3. Top-Football极宽足彩专业模块

Top-Football 极宽足彩专业模块库是专门针对足彩行业开发的函数库，各个模块文件均采用 TFB 字母作为文件开头，如图 B-3 所示是 Top-Football 极宽足彩专业模块库的主要模块组成图。



图 B-3　Top-Football 极宽足彩模块库组成图

Top-Football 极宽足彩专业模块库包括以下模块。

- tfb_main：足彩主程序入口模块。
- tfb_sys：足彩系统模块库，足彩系统所需的类定义，全局参数和变量，缩写为 tfsys。
- tfb_tools：足彩系统常用工具函数库，缩写为 tft。
- tfb_draw：足彩系统常用绘图函数库，缩写为 tfdr。
- tfb_data：足彩系统数据分析、预处理函数库，缩写为 tfdat。
- tfb_strategy：足彩系统常用策略模块库，缩写为 tfsty。
- tfb_backtest：足彩常用回溯分析函数库，缩写为 tfbt。

为了方便大家使用与移植，我们特意增加了一个 main 主模块，定义了一个 main()主函数入口，类似 C 语言，程序入口一目了然。

需要说明的是，随着未来极宽各个应用软件和系统的升级和发展，各个应用模块的构成可能会有所变动，请大家注意。

4. TopQuant金融量化系统模块构成

Top-Football 极宽足彩专业模块库是专门针对足彩行业开发的函数库，也可以说是 TopQuant 极宽量化系统的足彩移植版本，如图 B-4 所示是 TopQuant 极宽量化系统模块构成图。

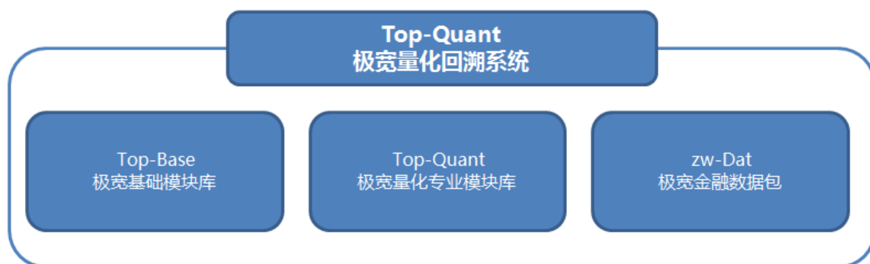


图 B-4　TopQuant 极宽量化系统模块构成图

由图 B-4 所示的 TopQuant 极宽量化系统模块构成图可以看出，TopQuant 量化系统与 TFB 足彩系统结构基本相同，只是数据源和专业应用模块不同，底层的 Top-Base 极宽基础模块库更是完全相同。

如图 B-5 所示是 TopQuant 极宽量化专业模块库的构成图。



图 B-5　TopQuant 极宽量化专业模块库构成图

TopQuant 极宽量化专业模块库包括以下模块。

- tq_main：极宽量化分析主程序入口模块。
- tq_sys：量化系统模块库，系统所需的类定义，全局参数和变量，缩写为 tqsys。

- tq_tools：量化常用工具函数库，缩写为 tqt。
- tq_draw：量化常用绘图函数库，缩写为 tqdr。
- tq_data：量化数据分析、预处理函数库，缩写为 tqdat。
- tq_strategy：量化系统常用策略模块库，缩写为 tqsty。
- tq_backtest：量化常用回溯分析函数库，缩写为 tqbt。

需要说明的是，随着未来极宽各个应用软件和系统的升级和发展，各个应用模块的构成可能会有所变动，请大家注意。

为了方便大家使用与移植，我们特意增加了一个 main 主模块，定义了一个 main()主函数入口，类似 C 语言，程序入口一目了然。